User's Manual

User's Manual

TimeStorm®

Version 4.4.2

Embedded Linux from a Trusted Source

timesys®

Table of Contents

TimeStorm Overview

TimeStorm is an Eclipse-based integrated development environment (IDE) for embedded application development. TimeStorm 4.4.2 is based on Eclipse Kepler (Eclipse 4.3.0 and C/C++ Development Tools 8.2.0). TimeStorm supports:

- C/C++ application development
- Qt application development
- Kernel Development
- Kernel Module Development
- SDK Management
- Remote Target Management
- Remote Target Console
- Remote Run / Debug of applications
- Yocto SDK
- QEMU
- Remote System Explorer
- Gprof
- Gcov
- LTTng
- OProfile
- Valgrind

Installing TimeStorm

Host System Requirements

Hardware

You will need a 64-bit or 32-bit machine with a minimum of 1GB RAM and 1GB free disk space to run TimeStorm.

Operating System

You will need a 64-bit or 32-bit latest Linux distribution. TimeStorm 4.4.2 is tested on Ubuntu 14.04 and Fedora 20, however you should be able to run TimeStorm on any latest Linux distributions like Red Hat, Debian, CentOS or SUSE. If you just have a Windows machine, you can run TimeStorm in a virtual machine running Linux.

Software Packages

Native-platform application development: For building applications that run on the local host, install the compiler for building C/C++ applications by running:

```
Ubuntu: sudo apt-get install build-essential
Fedora: yum groupinstall "C Development Tools and Libraries"
```

Cross-platform application development: For building applications that run on a remote hardware target, install an SDK for the target platform which also includes the cross compiler. You can download and install a pre built SDK for the target platform or build an SDK on your local host and install it. TimeStorm supports both Timesys Factory SDKs and Yocto Project-based SDKs, so you can install the SDK of your choice.

Timesys Factory SDK: A Timesys Factory SDK includes the cross compiler, bootloader, kernel image, RFS and kernel sources for your target platform.

Download and install a Factory SDK: You can download a pre-built Factory SDK by visiting https://linuxlink.timesys.com/download and install it by following the instructions on the board- -specific SDK download page.

Build and install a Factory SDK: You can build an SDK for your target platform by using Timesys Factory and following the instructions in: https://linuxlink.timesys.com/docs/wiki/factory/FactoryGSG

Yocto SDK: Yocto does not provide a single installer for the cross compiler, kernel and RFS. They have to be downloaded separately.

Download and install a Yocto SDK: You can download a pre-built Yocto toolchain from http://downloads.yoctoproject.org/releases/yocto/yocto-1.6/toolchain/ and install it by following the instructions in: http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

Build and install a Yocto SDK: You can build a Yocto SDK on your host and install it by following the instructions in: http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

Additional Packages: If you will be working on kernel projects where you will need to customize and build a Linux kernel, you will use either menuconfig or xconfig to configure the Linux kernel. You will need libncurses5-dev and xterm packages for menuconfig, and qt3-dev-tools or qt4-dev-tools package and xterm for xconfig.

Target Software Requirements

TimeStorm can use scp/ftp/nfs to download files and ssh/telnet/serial to communicate with the remote hardware target. Pre-built Timesys Factory SDKs include the packages that are required for TimeStorm to communicate with the target. If you are building a custom SDK, choose your target communication choice from the list below and include the corresponding packages in your target RFS.

- **For ftp:** inetutils, proftpd, pure-ftpd, vsftpd, ncftp
- **For telnetd:** busybox, inetutils, netkit-telnet
- **For scp:** dropbear, openssh
- **For ssh:** dropbeah or openssh
- **For sftp:** openssh
- **For nfs:** NFS enabled Kernel and busybox

** For tools that include uploading files from the remote target to host (OProfile, LTTng, Valgrind) you should have openssh server running on the target. Dropbear does not include sftp and does not work for uploading files.

Download TimeStorm

You can download TimeStorm by visiting: https://linuxlink.timesys.com/ide . You can download a 64-bit or 32-bit tarball depending on your operating system. To check whether you are running a 32-bit or 64-bit OS, run `uname -m` in a terminal. If the response is `i686,` you are running a 32-bit OS. If the response is `x86_64,` you are running a 64-bit OS.

Install TimeStorm

To install TimeStorm, simply un-tar the tarball. You can un-tar the tarball using the GUI, or by running `tar -zxf timestorm-full-install-4.4.2.<build>-<..>.tgz.` TimeStorm will be installed in `timestorm-4.4.2` folder.

Create a License

TimeStorm 4.4.2 uses a node-locked license which is a text files with .lic extension. The node-locked license enables TimeStorm features on one machine only and is restricted to the MAC address of the machine. Once a license is installed and used, it cannot be moved to a machine with a different MAC address. To use TimeStorm on another machine, you have to create and install another license for the machine.

To create a license for TimeStorm, you must have a LinuxLink seat assigned to you.

- You can create a new license for TimeStorm three times within the life of the subscription.
- If you need to create additional licenses beyond this limit, contact Timesys. These requests are handled on a case-by-case basis.
- ***Manager* —** If you are the Manager of your LinuxLink account, you can create a license for all the seats within your account.
- ***Developer*** — If you are a Developer, you can create licenses for the seats assigned to you.

To create a TimeStorm license:

1. Note the MAC address of the machine on which you will be using TimeStorm. You can view the MAC address by running `ifconfig`. You may use the MAC address of any of the active network interfaces. If you are running TimeStorm in a Virtual Box environment, make sure you use the MAC address of the NIC inside the virtual machine.
2. Log into your LinuxLink account https://linuxlink.timesys.com/, and then:
   - If you are a ***Manager***, click on the account name at the top of the page, and click on a username link in the Members section.
   - If you are a ***Developer,*** click on your user name at the top of the page.
3. Scroll to the bottom of the page, and click the 'Edit Licenses' button in the Active Licenses section.
4. Click the 'Create' button located to the right of the seat for which you want to create the license.

- Enter the MAC address and a descriptive name for the license. It can be helpful to include the type of operating system and computer that will use the license.
- The license expiration date and the user’s email address are entered automatically.
- Click ‘Create’ to generate the license. The license is created, emailed to the user and displayed in the web page.

Install License

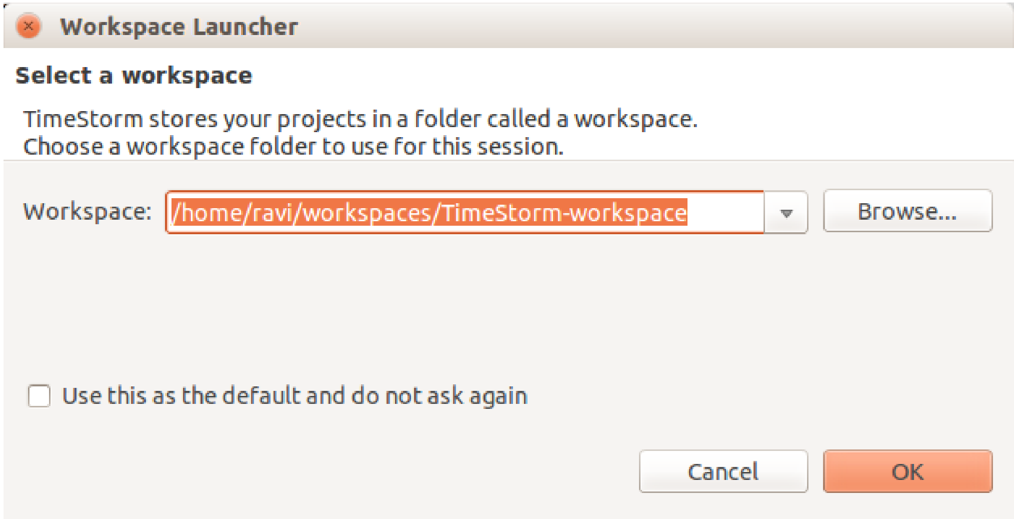
Installing a license involves copying your license file into the appropriate directory on the TimeStorm host. You can obtain the license online by visiting https://linuxlink.timesys.com/user/edit/licenses.xhtml or from the attachment of the email sent to you. Save the license file in one of these two locations:

1. **<user home directory>/.timesys/timestorm/licenses**
   The TimeStorm user's home directory is typically something like `/home/<username>` on Linux systems.
   - Licenses copied to this location will work for this user for all version-compatible TimeStorm installations on this computer.
2. **<TimeStorm installation directory>/licenses**
   TimeStorm checks for a directory named ‘licenses’ that is located within the TimeStorm installation directory.
   - This licenses directory is created automatically when TimeStorm is installed. Licenses installed in this directory will work for this installation of TimeStorm only.

Run TimeStorm

To run TimeStorm, `cd` to the TimeStorm installation directory and run `./timestorm`

TimeStorm will launch and prompt you for the workspace to use, a location on your file system to store all your application projects and related information, as shown in *figure 1*.



Workspace Launcher

Select a workspace

TimeStorm stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

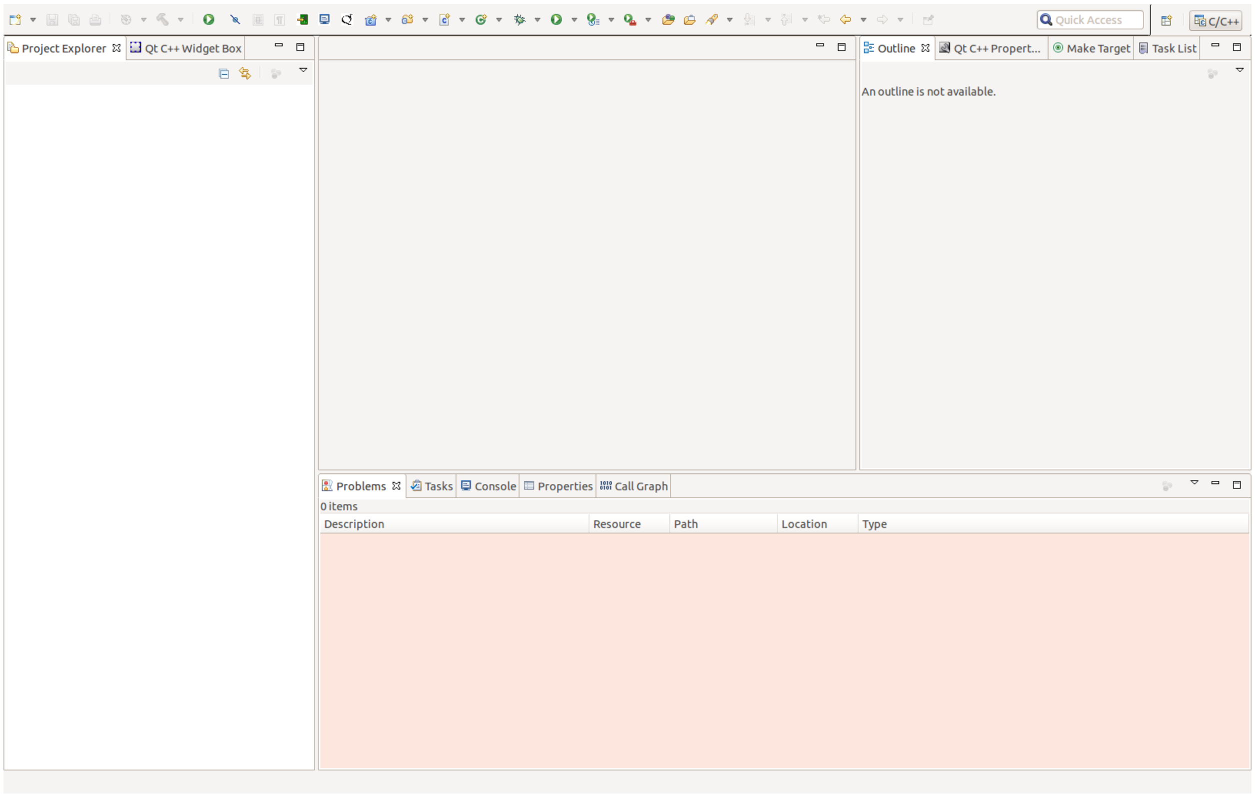
Workspace: /home/ravi/workspaces/TimeStorm-workspace

□ Use this as the default and do not ask again

Figure 1: Workspace Launcher

The default workspace location is `/home/<username>/workspaces/TimeStorm-workspace`. You may leave this unchanged. To avoid being prompted for the workspace every time TimeStorm is run, you can select the ‘Use this as the default and do not ask again’ checkbox.

Click OK to continue, and you will see TimeStorm running as shown in *figure 2*.



An outline is not available.

0 items

Figure 2: TimeStorm Initial Launch

Documentation

Documentation for TimeStorm and Eclipse can be viewed by clicking Help > Help Contents. If you are new to Eclipse, we recommend you to read the first chapter, “Workbench User Guide” to get used to TimeStorm terminology and “C/C++ Development User Guide” to get used to C/C++ project development.

Embedded Development with TimeStorm

This chapter explains the various steps involved in embedded development using TimeStorm:

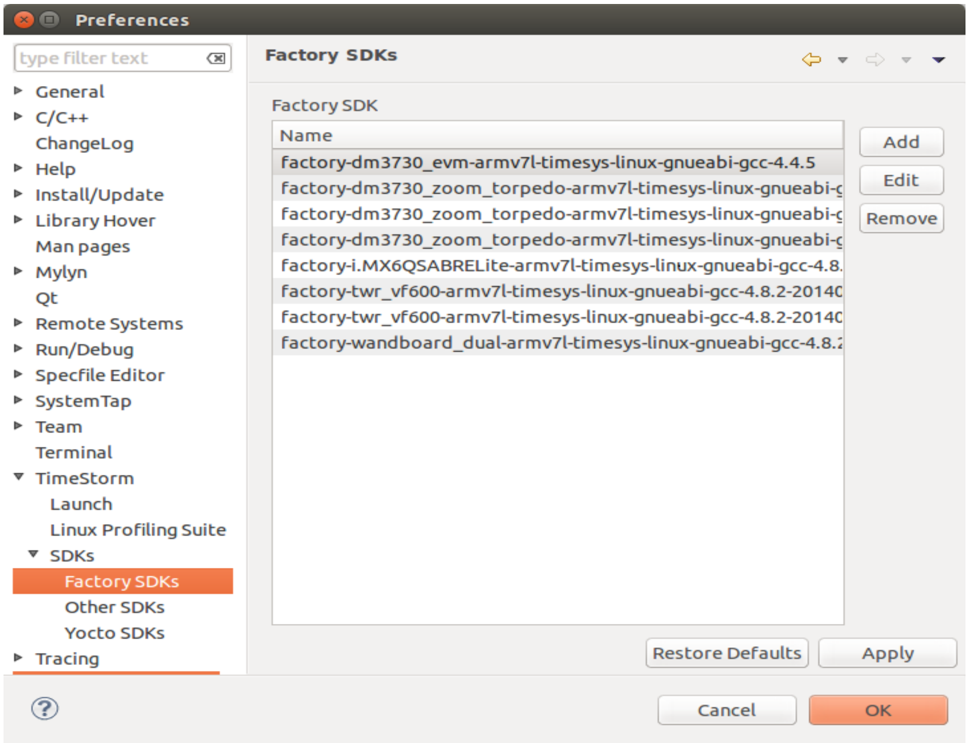
- Setup the application development environment by installing SDK
- Create an application project
- Build the application
- Run the application on the remote target
- Debug the application that is running on the target
- Profile the application on the remote target
- Integrate the application into your build system
- Share the application code by saving it in a version control system

Setup

Setup includes installing an SDK that includes a cross compiler for building applications. TimeStorm supports Factory, Yocto or any other SDKs. You can download a pre-built SDK and install it or locally build an SDK on your host and install it. The pointers to Factory and Yocto SDK are listed here.

Detected SDKs

TimeStorm automatically detects Factory and Yocto SDKs installed in a default location. The detected SDKs can be viewed by clicking Window > Preferences > TimeStorm > SDKs > Factory / Yocto / Other SDKs. *Figure 3* shows the factory SDKs detected by TimeStorm.



Factory SDKs

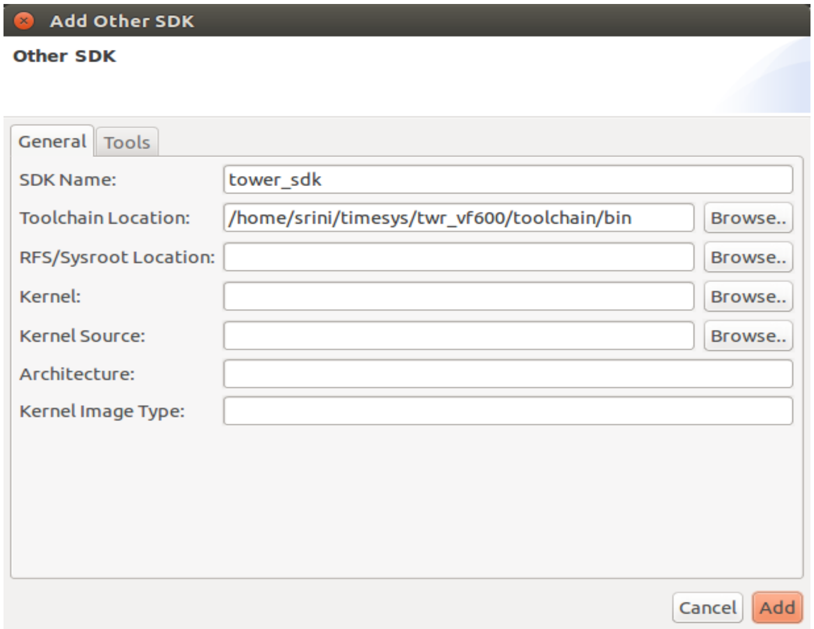
Factory SDK

Name
factory-dm3730_evm-armv7l-timesys-linux-gnueabi-gcc-4.4.5
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-g
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-g
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-g
factory-i.MX6QSABRELite-armv7l-timesys-linux-gnueabi-gcc-4.8.
factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-2014C
factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-2014C
factory-wandboard_dual-armv7l-timesys-linux-gnueabi-gcc-4.8.2

Figure 3: Factory SDKs

Adding an SDK

If your SDK is not detected, you can manually add the SDK to TimeStorm by selecting Factory / Yocto / Other SDK and clicking the Add button (shown in *figure 3*).



Other SDK

SDK Name: tower_sdk

Toolchain Location: /home/srini/timesys/twr_vf600/toolchain/bin

RFS/Sysroot Location:

Kernel:

Kernel Source:

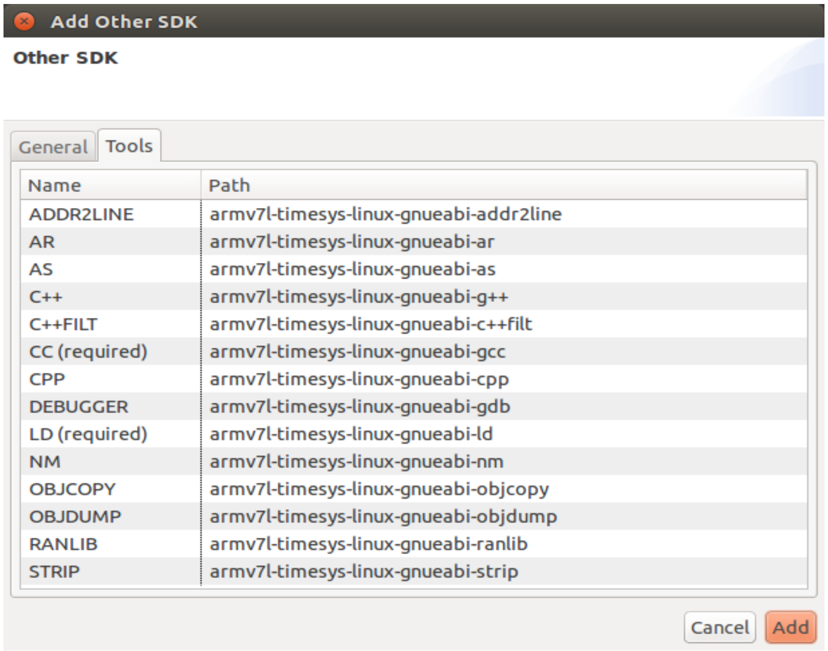
Architecture:

Kernel Image Type:

Figure 4: Adding an SDK

In the Add SDK dialog shown in *figure 4*, enter SDK Name and locate the toolchain binary files folder. RFS/Sysroot Location, Kernel, Kernel Source, Architecture and Kernel Image Type are optional. Kernel Source, Architecture and Kernel Image type values are required for working with a Kernel Project. RFS/Sysroot Location and Kernel values are required for launching QEMU in TimeStorm. Note that TimeStorm supports QEMU only with Yocto SDKs.

Tools tab in *figure 5* shows the list of tools and their path that have been detected in the specified toolchain directory. You can edit the path of the tools. Toolchains using gcc with a cross-compiler prefix will be automatically detected.



Name	Path
ADDR2LINE	armv7l-timesys-linux-gnueabi-addr2line
AR	armv7l-timesys-linux-gnueabi-ar
AS	armv7l-timesys-linux-gnueabi-as
C++	armv7l-timesys-linux-gnueabi-g++
C++FILT	armv7l-timesys-linux-gnueabi-c++filt
CC (required)	armv7l-timesys-linux-gnueabi-gcc
CPP	armv7l-timesys-linux-gnueabi-cpp
DEBUGGER	armv7l-timesys-linux-gnueabi-gdb
LD (required)	armv7l-timesys-linux-gnueabi-ld
NM	armv7l-timesys-linux-gnueabi-nm
OBJCOPY	armv7l-timesys-linux-gnueabi-objcopy
OBJDUMP	armv7l-timesys-linux-gnueabi-objdump
RANLIB	armv7l-timesys-linux-gnueabi-ranlib
STRIP	armv7l-timesys-linux-gnueabi-strip

Figure 5: SDK - Detected Tools

Click Add to add the SDK.

Automatically detected and manually added SDKs are displayed in a drop down list in new project wizards, build settings and launch configuration dialogs. User can choose an SDK from the list, and select or change the SDK used with an application.

Create

Application projects are created by using the New C or C++ Project wizard. TimeStorm uses the existing CDT C and C++ Project wizards, but adds the ability to select an SDK to use to build the project.

Create New Project

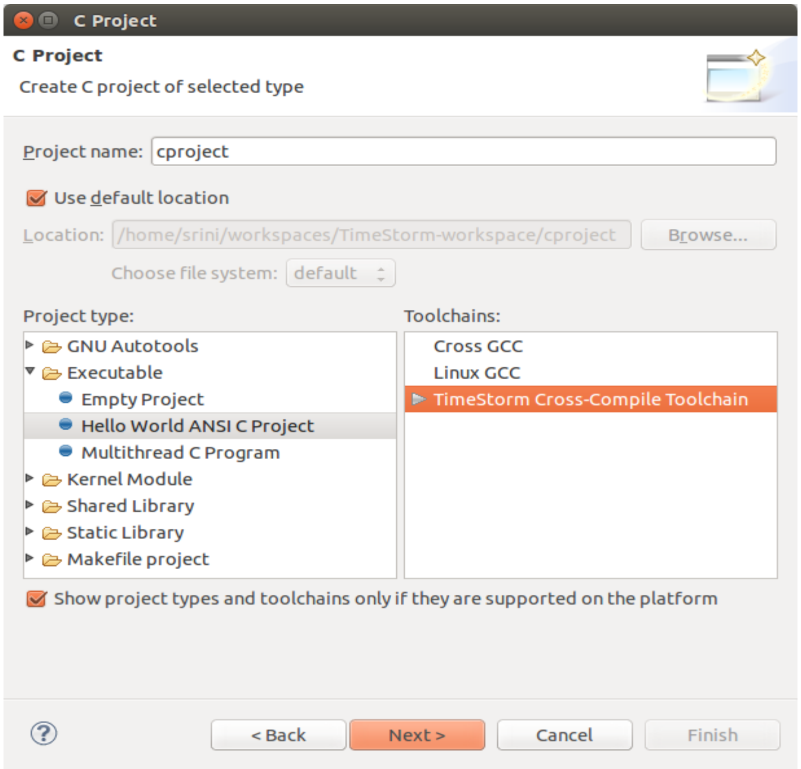
To create a new application project:

Open the 'New Project' wizard by choosing *File > New > Project* from the main menu OR click the 'New' button and use the drop-down menu to the right of the button to choose 'Project'.

Expand C/C++ , choose a C or C++ Project and click Next. The new C/C++ Wizard *(shown in figure 6)* can be used to create several different types of projects, including 'Executable' and 'Shared and Static Library'.

Project page

1. Enter a Project Name.
2. Under the 'Project Type' heading, expand Executable and select the initial source code (empty, Hello World or Multithread) you would want in the project.
3. Under the Toolchains heading select the TimeStorm Cross-Compile Toolchain.
4. Click 'Next' to bring up the 'Basic Settings' page *(shown in figure 7)*.



C Project

C Project

Create C project of selected type

Project name: cproject

☑ Use default location

Location: /home/srini/workspaces/TimeStorm-workspace/cproject

Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Multithread C Program
- Kernel Module
- Shared Library
- Static Library
- Makefile project

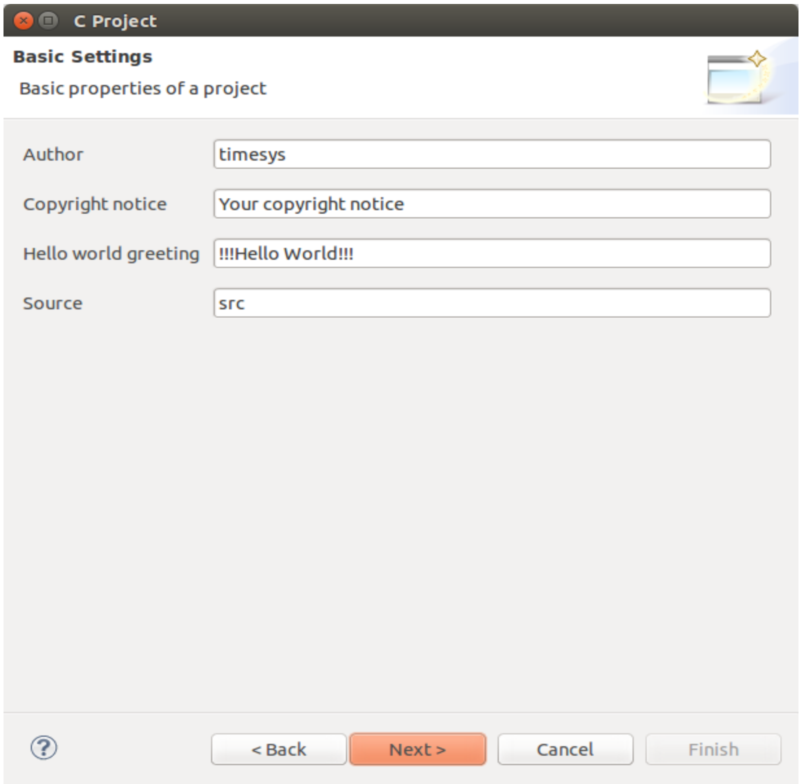
Toolchains:
- Cross GCC
- Linux GCC
- TimeStorm Cross-Compile Toolchain

☑ Show project types and toolchains only if they are supported on the platform

Figure 6: New C/C++ Project Wizard

Basic settings page

In this page, set the basic properties of the project like Author, Copyright notice, Greeting message and Source (where the project source files are to be stored within project). Click 'Next' to bring up the 'Select Configurations' page (*shown in figure 8)*.



Author: timesys
Copyright notice: Your copyright notice
Hello world greeting: !!!Hello World!!!
Source: src

Figure 7: Basic Settings

Select build configuration page

Build Configuration is a set of pre-defined compiler and linker settings that are used to build a project. By default, TimeStorm creates three Build Configurations for new projects:

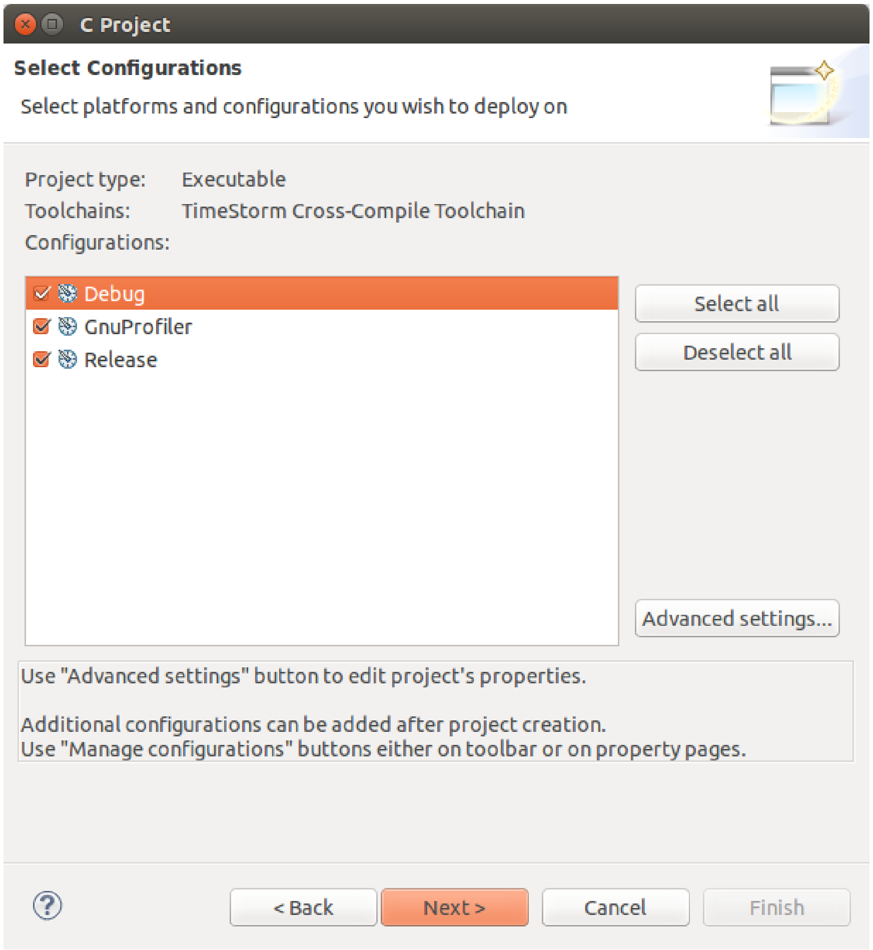
Debug — This setting performs no code optimization and attaches complete debugging information. These settings are designed make debugging as easy as possible.

Release — This setting has the highest compiler code optimization settings. The output will not include any debugging symbols. Code compiled with this configuration is ready for production use.

GnuProfiler — This setting builds the software with debugging information as well as code to collect profiling information via GNU gprof.

You can change the build configuration settings before creating the project by clicking the 'Advanced Settings…' button. You can also change these settings after creating the project.

Click Next to continue.



C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable

Toolchains: TimeStorm Cross-Compile Toolchain

Configurations:

- ☑ Debug
- ☑ GnuProfiler
- ☑ Release

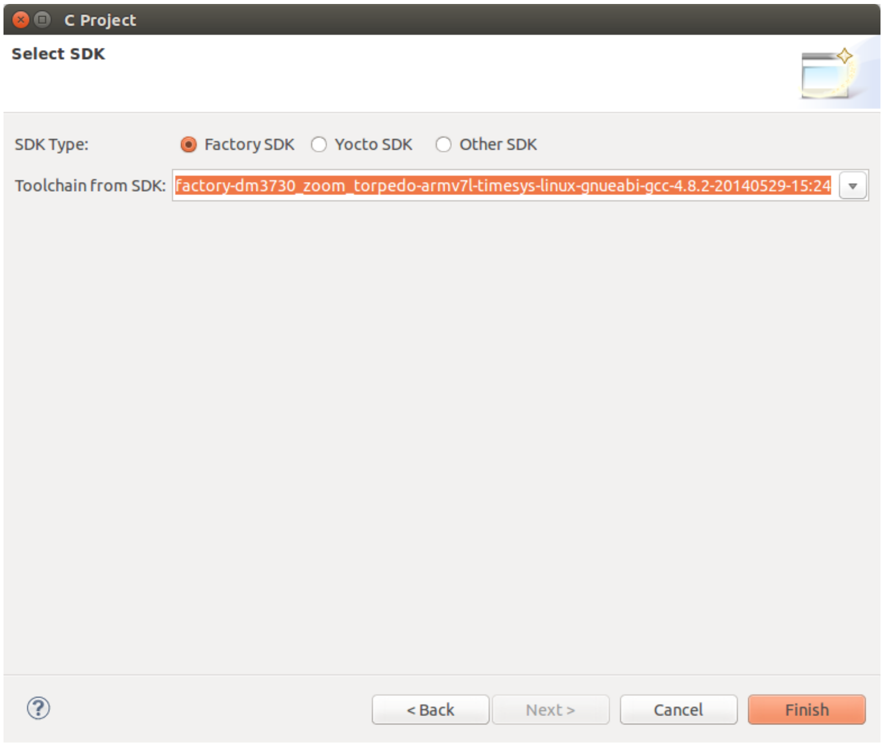
Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

Figure 8: Select Build Configuration

Select SDK page

The Select SDK page show in *figure 9* allows you to choose the SDK you want to use with the project. Select the SDK Type and then choose the SDK.



C Project

Select SDK

SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK: factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24

Figure 9: Select SDK

Click Finish to create and exit the wizard. The new project is created and displayed in the Project Explorer view.

Edit project

You can add more source folders, source files and header files to the project from the Project context menu, by right clicking on the project, clicking New and selecting your choice as shown in figure 10.

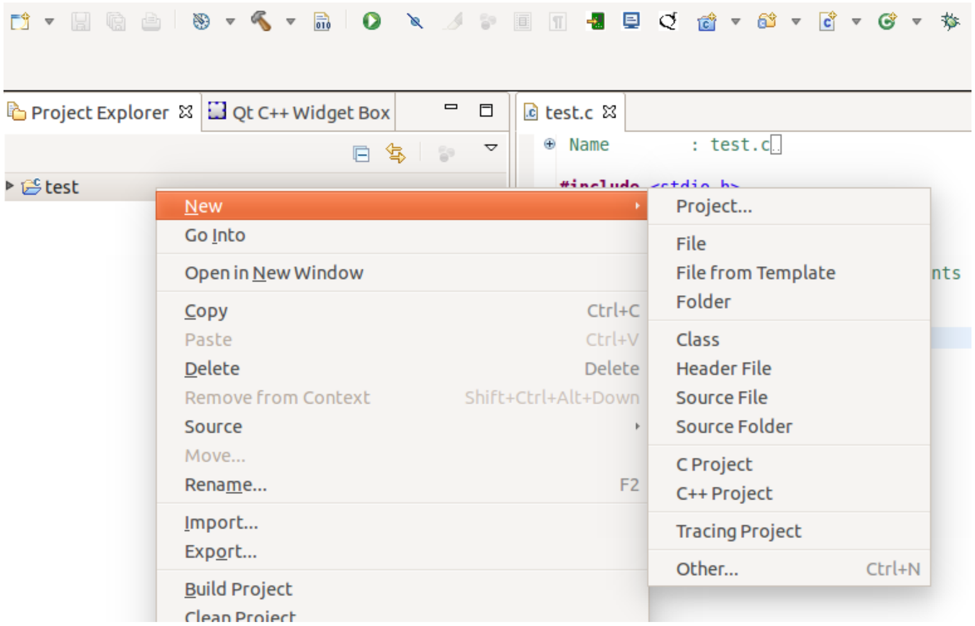


Figure 10: Project Context Menu

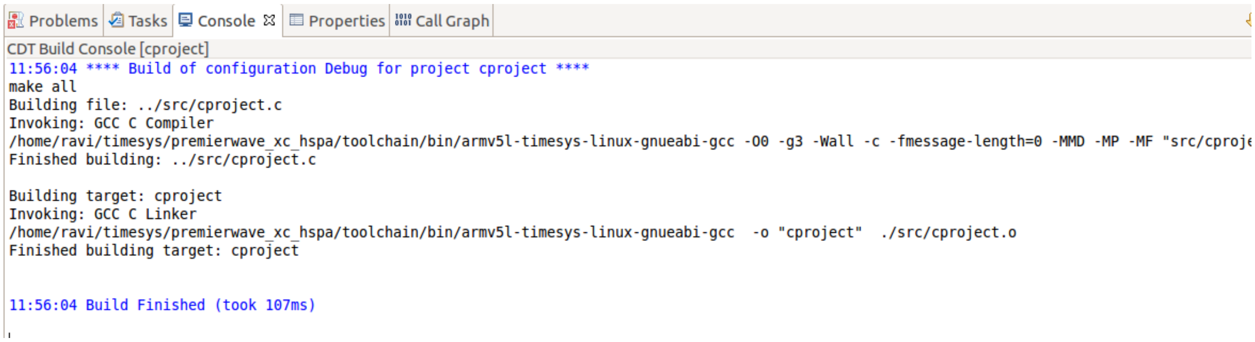
Static and Shared Libraries

Apart from executable, you can also create static and shared libraries using C/C++ project wizard. Static and shared libraries allow the user to place certain functionality outside of the main program, frequently so the same library can be shared across projects. Static libraries are incorporated by copying the bits into the main program at link time while shared libraries are linked into the program before it runs by a dynamic library loader. Shared libraries can also be accessed when the program is running by loading the libraries into memory and calling functions, without the linking step before the program runs. The table below shows a comparison between static and shared libraries.

	Library Type	
Feature	Static	Shared
Output File Name	lib<project name>.a	Lib<project name>.so
Can Update at Run-time	No	Yes
Dynamic Loader Needed	No	Yes
Code Shared Across Applications	No	Yes

Build

You can build the project by clicking on Build Project in the project context menu. The build progress is displayed in the Console view as shown in *figure 11.*



```
11:56:04 **** Build of configuration Debug for project cproject ****
make all
Building file: ../src/cproject.c
Invoking: GCC C Compiler
/home/ravi/timesys/premierwave_xc_hspa/toolchain/bin/armv5l-timesys-linux-gnueabi-gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF "src/cproje
Finished building: ../src/cproject.c

Building target: cproject
Invoking: GCC C Linker
/home/ravi/timesys/premierwave_xc_hspa/toolchain/bin/armv5l-timesys-linux-gnueabi-gcc  -o "cproject"  ./src/cproject.o
Finished building target: cproject


11:56:04 Build Finished (took 107ms)
```

Figure 11: Build progress in console view

If the build fails, the build errors can be viewed in the Problems view.

To fix the build errors, to change the build settings or to change the SDK used to build the project, click on Properties in the project context menu, and click on C/C++ Build > Settings.

How TimeStorm Builds a Project

TimeStorm uses Build Configurations to control the project build process. When creating a project using a wizard, three build configurations will be created by TimeStorm and associated with the project. One of these build configurations is the “active” configuration (as illustrated by the star in the illustration, below) and that will be used by default when creating a build. Any build configuration can be selected as the active configuration, as this concept exists so the user does not need to select which build settings to use when creating a build.

Figure 12, below, shows the relationship between the project, build configurations, SDKs and the make files generated by TimeStorm.

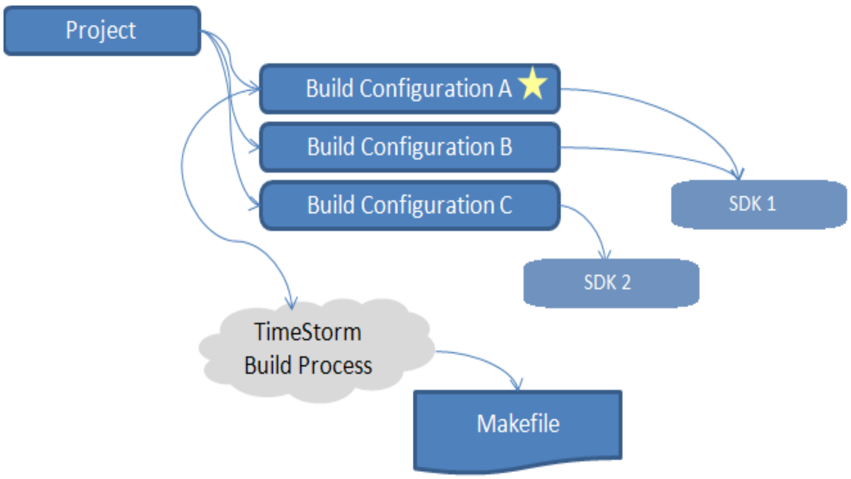


Figure 12: Relationship between Project, Build Configuration, SDK and Makefile

The following describes how TimeStorm uses the makefile and build config to build the project. TimeStorm will use the default build configuration, following these steps:

1. **Create a directory with the name of the build configuration** — The output for the build is stored under a directory that matches the name of the build configuration used for the build. If this directory does not exist, it will be created. TimeStorm must have the ability to create directories in the workspace or this operation will result in an error.

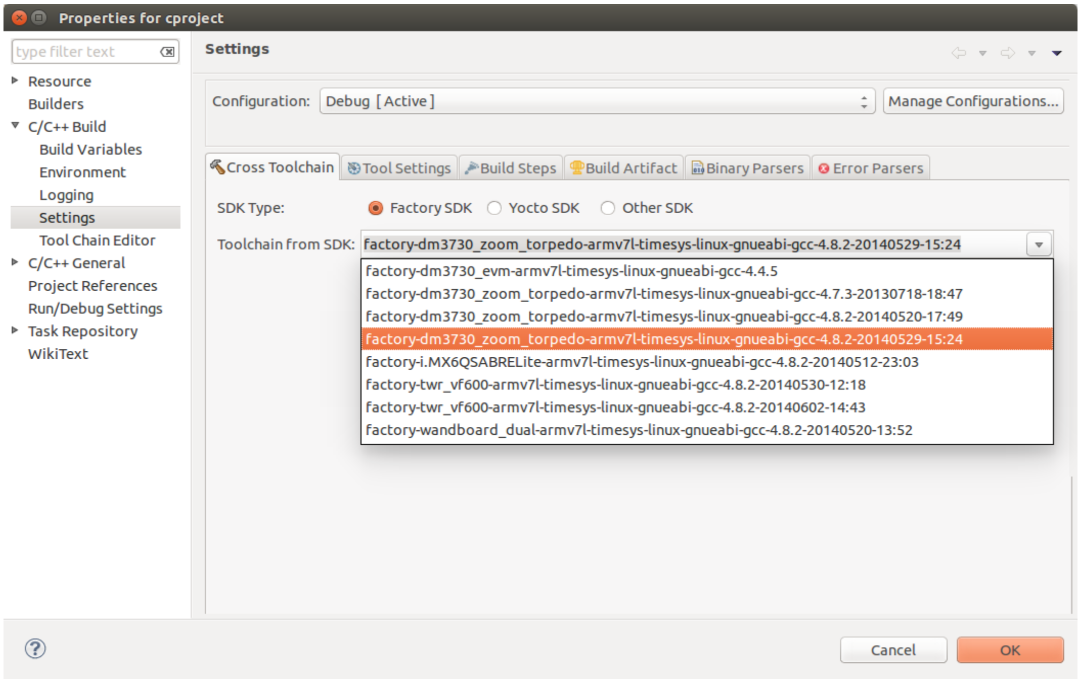
2. **Scan the project, create a makefile that invokes the appropriate build program for the source file** — Before the build process occurs, TimeStorm will scan the project for files it knows how to build based on extension, ignoring files that it does not know how to handle. For those files that it knows how to process, TimeStorm will then examine the files, calculate dependencies, and emit a standard GNU makefile.

3. **Execute the makefile, display results in the Console view** — After creating the makefile, TimeStorm uses GNU to perform the build. The results of the build process are displayed in the Console view.

4. **Scan the results, create markers for errors and warnings** — After the build, the results are scanned and TimeStorm translates errors and warnings for files into markers that appear next to the offending line and in the problems view.

Changing project SDK

You can change the SDK used to build the project by clicking the Cross Toolchain tab as shown in *figure 13*.



Configuration: Debug [Active]

SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK: factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24

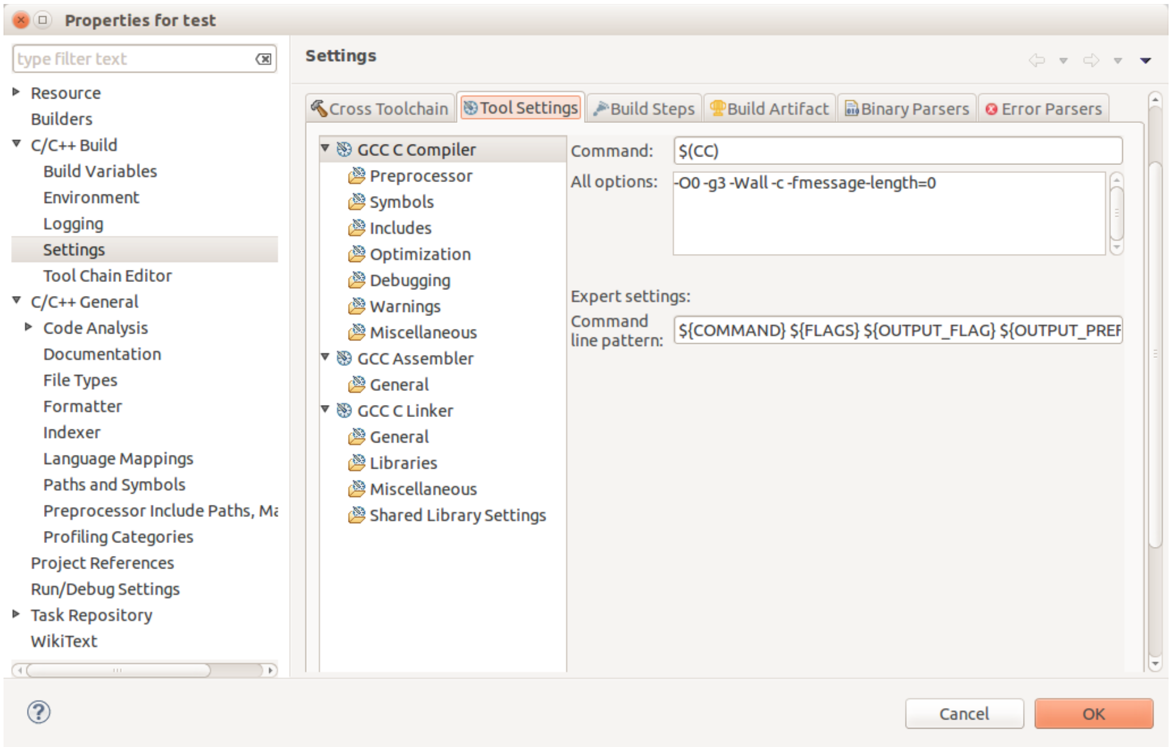
```
factory-dm3730_evm-armv7l-timesys-linux-gnueabi-gcc-4.4.5
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.7.3-20130718-18:47
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140520-17:49
factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24
factory-i.MX6QSABRELite-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140512-23:03
factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140530-12:18
factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140602-14:43
factory-wandboard_dual-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140520-13:52
```

Figure 13: Changing the SDK for an existing project

Note that the SDK is changed for the configuration displayed at the top of the Settings page. You can change the active configuration by clicking the Manage Configurations button. Be sure to clean the project, before building with the newly selected SDK.

Changing Compiler and Linker Settings

You can change the compiler and linker settings by clicking on the Tool Settings tab as shown in *figure 14.*



Command: $(CC)

All options: -O0 -g3 -Wall -c -fmessage-length=0

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREF

Figure 14: Project compiler and linker settings

Building Related Projects in a Workspace

In TimeStorm, projects are not hierarchical. This is much different than other systems where a project is frequently structured as a top-level directory with a directory for each component, with those directories nesting downward.

For example:

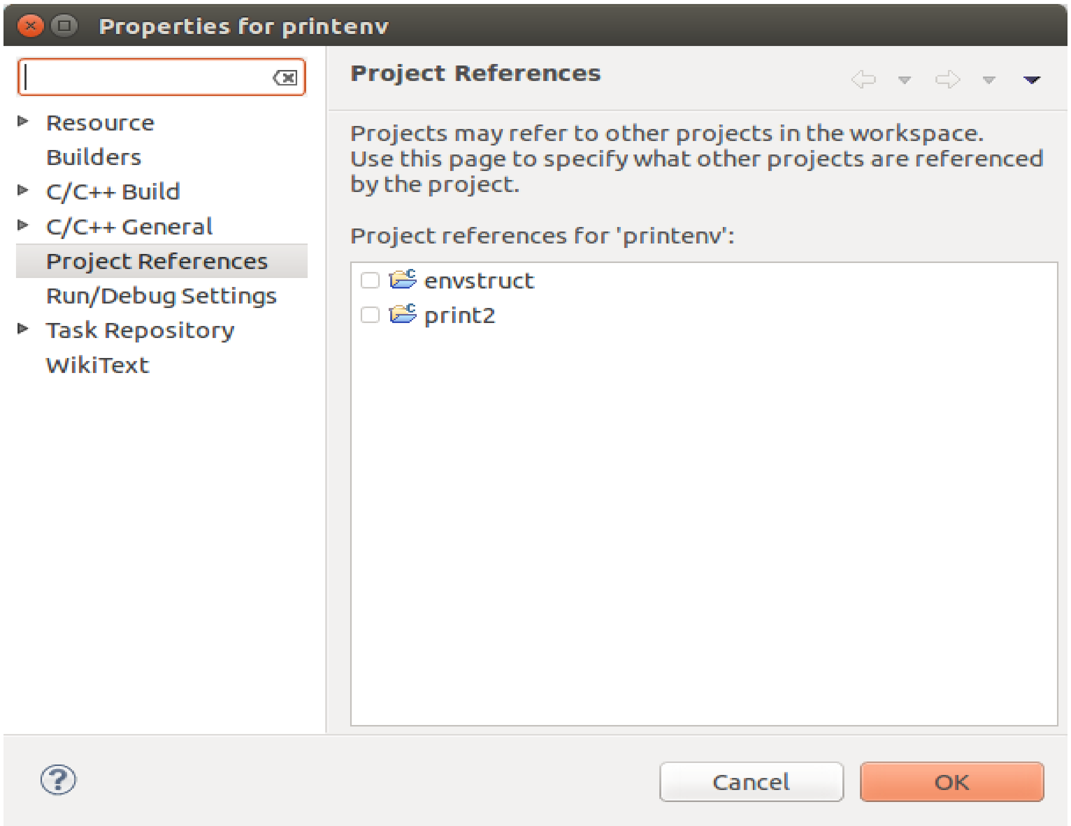
```
top-level-dir/
       application
       shared-lib-1/
              static-lib-1/
              static-lib-2/
       shared-lib-2/
              static-lib-1/
              static-lib-2/
```

In this project, the user would typically write a make file that built the projects in the following order:

```
shared-lib-2/static-lib-2/
shared-lib-2/static-lib-1/
shared-lib-1/static-lib-2/
shared-lib-1/static-lib-1/
application/
```

In TimeStorm, since all of the projects are peers of each other, enforcing the build order would occur through ‘Project References’, which is part of the project properties. To access the Project References:

1. Open the 'Project Properties' dialog by right-clicking and selecting 'Properties.'
2. Select the 'Project References' entry on the left. The dialog will look like the following, shown in *Figure 15.*



Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for 'printenv':

□ envstruct
□ print2

Figure 15: Project References Panel

By selecting entries in this dialog, TimeStorm will build those projects before the current project (if, in fact, they needed to be rebuilt). Project references can be nested several levels deep, so that if a referenced project has other references, those references also will be rebuilt, if necessary. This ensures that the top-level project has all of its dependencies built before starting its build. In this way, the developer has the same degree of control as one would have when using a traditional "nested" make file.

Run

Once the application is successfully built, you can run it on the remote hardware target. To run the application, you have to define a hardware target and create a run configuration.

Hardware Targets Window

The Hardware Targets window enables you to:

- Add a target to the list of registered targets.
- Edit the information for a registered target.
- Delete a target from the list of registered targets.
- Check connectivity from the host to a target.

You can access the Hardware Targets window either by clicking the Hardware Targets toolbar button or by selecting *Run > Hardware Targets* from the main menu, as shown in *Figure 16*, below.

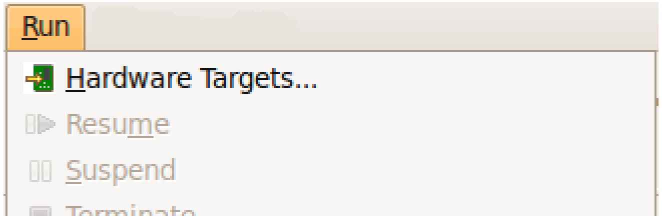


Figure 16: Hardware Targets Menu

Setting up a target requires information for the following three tabs:

TAB 1: Download — How to download files to the target *(shown in Figure 17),*

TAB 2: Execute — How to communicate to execute on the target *(shown in Figure 18)* and

TAB 3: Raw Log — How to test the connectivity between the host and the target *(shown in Figure 19)*.

TAB 1: Download

TimeStorm allows files to be downloaded to the target in a few different methods. You will be prompted to select one of the options below.

FTP – Files will be downloaded from the host to the target using FTP.

SCP – Files will be downloaded from the host to the target using SCP.

NFS – TimeStorm will copy the files locally in the target Root File System (RFS) mounted over NFS.

None – The host will be connected to the target, but no data will be downloaded. You can then do any needed downloading using the console.

FTP/SCP Settings

The corresponding fields for both the FTP and SCP options *(shown in Figure 17)* are as follows:

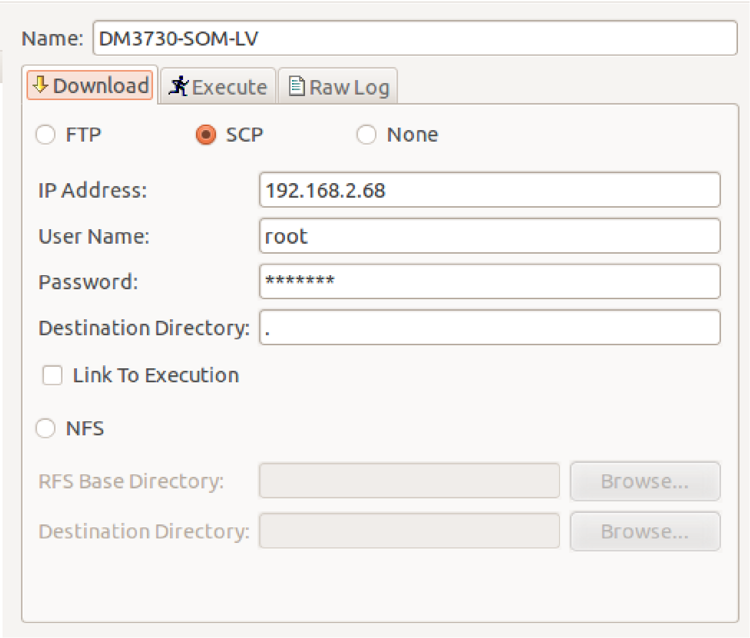
IP Address — Enter either the IP address or the hostname of the target. If you use a hostname, your network must include access to a Domain Name Server (DNS) lookup facility.

User Name — Enter the user name for FTP or SCP login on the target, as appropriate.

Password — Enter a non-blank password for FTP or SCP login, as appropriate.

Destination Directory — Enter the path to which the files will be transferred on the target. FTP settings sometimes restrict the location of files copied to a directory under the home directory of the user. Therefore, the actual destination directory depends on the configuration of your FTP software. In some cases, directories specified in this panel are appended to the user’s home directory on the target. Ensure the user name that you specified has read/write permissions to the destination directory.

Link to Execution — When this checkbox is selected, all of the information listed in the fields corresponding to the FTP or SCP option is automatically entered into the fields in the Execute tab. Those fields are not editable in the ‘Execute’ tab when this checkbox is selected. This checkbox is selected by default. When this checkbox is deselected, you must manually enter the corresponding information in the fields in the ‘Execute’ tab.



Name: DM3730-SOM-LV

○ FTP ● SCP ○ None

IP Address: 192.168.2.68

User Name: root

Password: *******

Destination Directory: .

□ Link To Execution

○ NFS

RFS Base Directory:

Destination Directory:

Figure 17: Download Tab

NFS Settings

The corresponding fields for the NFS option *(shown in Figure 17)* are as follows:

***RFS Base Directory*—** Use the Browse button to select the location of the RFS on the host. This location is where the target's root filesystem is mounted over NFS; for example, `/home/user/timesys/boardname/rfs`, where *user* is the current username and *boardname* is the name of the target board.

***Destination Directory* —** Use the Browse button to select the location in the RFS to where the files will be downloaded. The directory must be within the RFS Base Directory location that was selected. Ensure that you have read/write permissions to the destination directory.

After you have entered the required information in this tab, select the 'Execute' tab, and enter the appropriate information.

TAB 2: Execute

Use the Execute tab *(shown in Figure 18)* to specify the communication method that TimeStorm uses between the host and the target.

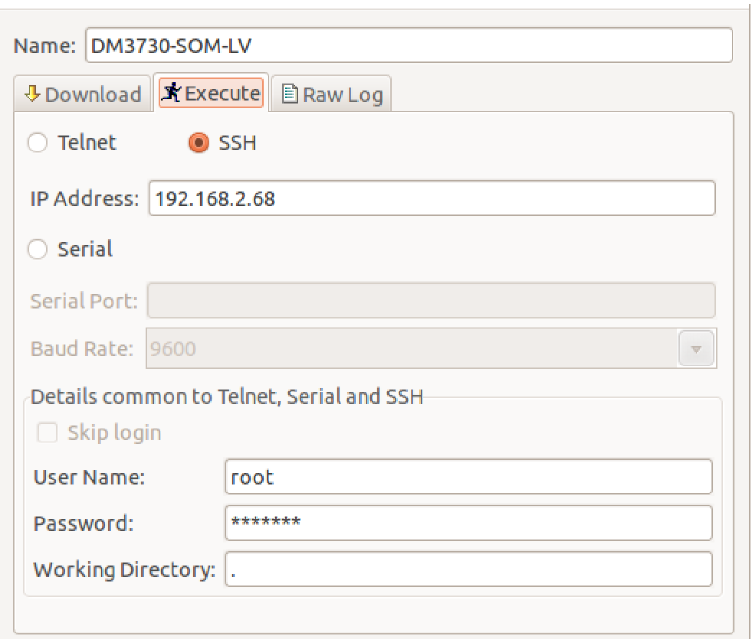
Within this tab, you can select from the following communication methods between the host and the target:

Telnet – Communication between the host and the target will occur when using Telnet.

SSH – Communication between the host and the target will occur when using secure shell (SSH).

Serial – Communication between the host and the target will occur using the serial connection.

You must fill in all of the corresponding fields for the method that you choose. TimeStorm answers the login and password prompts presented by the Telnet or SSH server on the target, based on the data that you enter in this tab.



Name: DM3730-SOM-LV

○ Telnet ● SSH

IP Address: 192.168.2.68

○ Serial

Serial Port:

Baud Rate: 9600

Details common to Telnet, Serial and SSH

□ Skip login

User Name: root

Password: *******

Working Directory: .

Figure 18: Execute Tab

Telnet/SSH Settings

The corresponding field for both the Telnet and SSH options are as follows:

***IP Address* —** Enter either the IP address or the hostname of the target. If you use a hostname, your network must include access to a DNS lookup facility.

The execution IP Address field is also used when connecting to gdbserver for TCP-based remote debugging. This field must be provided regardless of connection method to use TCP-based debugging.

Serial Settings

The corresponding fields for the Serial option are as follows:

***Skip login* —** If you are connecting directly to the target's bootloader, and thus will not get a login prompt, select this option. The rest of the common details will be grayed out.

***Serial Port* —** Enter the host's serial port; for example, /dev/ttyS0. If you use this option, be sure that the user that you specified has the proper permissions to access the serial port.

***Baud Rate*—** Use the drop-down list to select the appropriate baud rate for serial communication. The default rate is 9600 bps.

Common Settings for All Options

The common fields for all three options in the Execute tab are as follows:

***User Name* —** Enter your user name for a Telnet, SSH, or serial login, as appropriate, on the target.

***Password* —** Enter a non-blank password for the Telnet, SSH, or serial login, as appropriate, on the target. Note that the Timesys pre-built SDK does not set the root password in the RFS. You have to set the password for the root user after the target boots, and enter the password in this text field.

***Working Directory* —** Enter the directory to which control will be transferred after TimeStorm logs in to the target. By default, this is the same path to which the data will be transferred on the target. This directory must already exist.

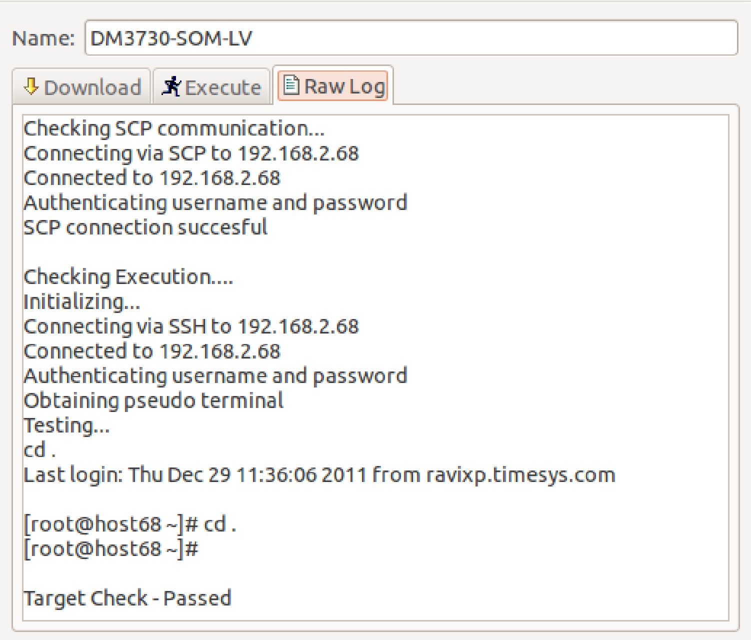
NOTE: *If the ‘Link to Execution’ checkbox is selected in the Download tab, the information in the fields corresponding to the FTP or SCP option in the Download tab is automatically entered into the common fields and the IP Address field in the Execute tab. In this case, the fields are not editable as long as the ‘Link to Execution’ checkbox is selected in the Download tab.*

TAB 3: Raw Log

After you register a target, you can check the connectivity to that target at any time. Each time you add a target or edit information for a target, Timesys recommends that you perform a connectivity check to verify that the host can communicate with the target. The target connectivity check does not occur automatically upon registering or applying changes to a target.

To verify connectivity to a target, select the target in the left panel of the ‘Targets’ window and click the ‘Check Link’ button. The Raw Log tab *(shown in Figure 9)* becomes active when the check starts.

If the host communicates successfully with the target, a ‘`Target Check – Passed`’ message *(shown in figure 19)* is displayed in the ‘Raw Log’ tab of the’ Targets’ window.

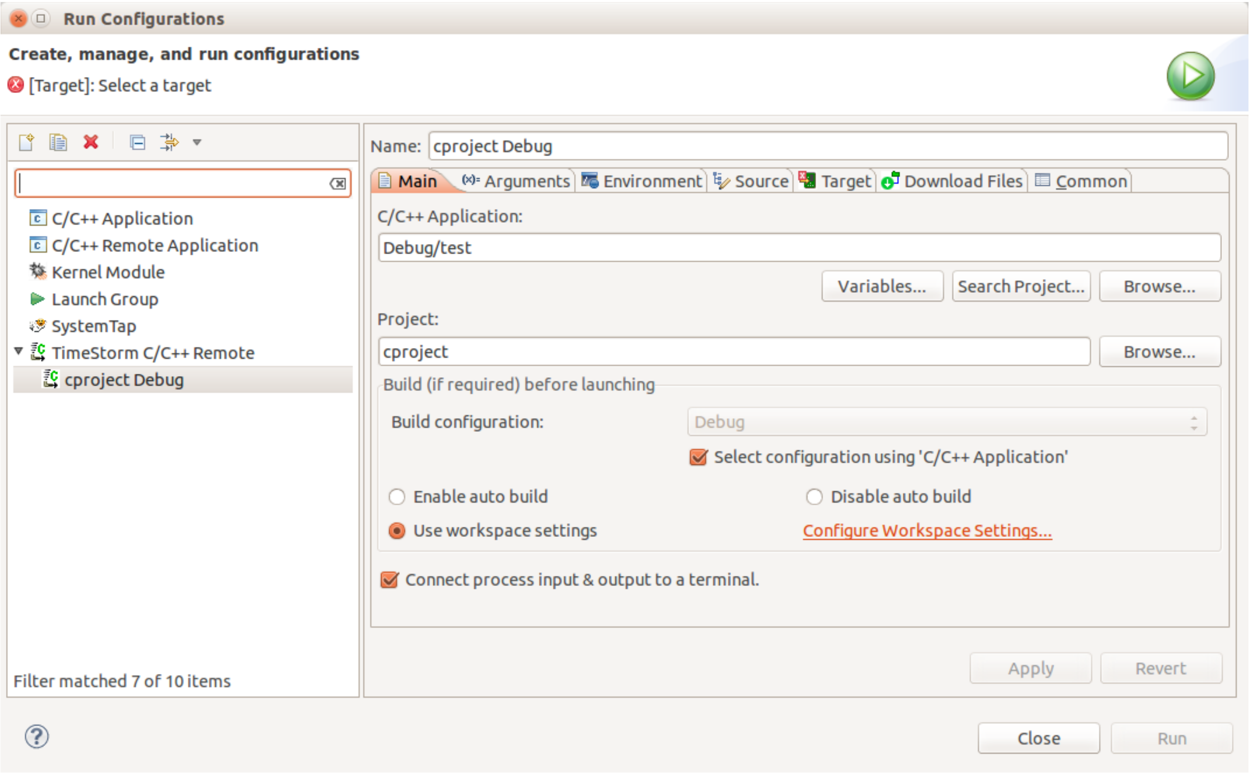


```
Checking SCP communication...
Connecting via SCP to 192.168.2.68
Connected to 192.168.2.68
Authenticating username and password
SCP connection succesful

Checking Execution....
Initializing...
Connecting via SSH to 192.168.2.68
Connected to 192.168.2.68
Authenticating username and password
Obtaining pseudo terminal
Testing...
cd .
Last login: Thu Dec 29 11:36:06 2011 from ravixp.timesys.com

[root@host68 ~]# cd .
[root@host68 ~]#

Target Check - Passed
```

Figure 19: Output from Check Link

Run Configuration

A Run Configuration is a collection of settings that are required to download and run an application. The run configuration is persistent, and can be launched multiple times to run the application. To run the application on the remote target, you have to create a run configuration. To create a run configuration, right click on the project and click on Run As > Remote C/C++ application. This will open the run configurations dialog as shown in *figure 20.*


Run Configurations

Create, manage, and run configurations

[Target]: Select a target

- C/C++ Application
- C/C++ Remote Application
- Kernel Module
- Launch Group
- SystemTap
- TimeStorm C/C++ Remote
  - cproject Debug

Filter matched 7 of 10 items

Name: cproject Debug

C/C++ Application: Debug/test

Project: cproject

Build (if required) before launching

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

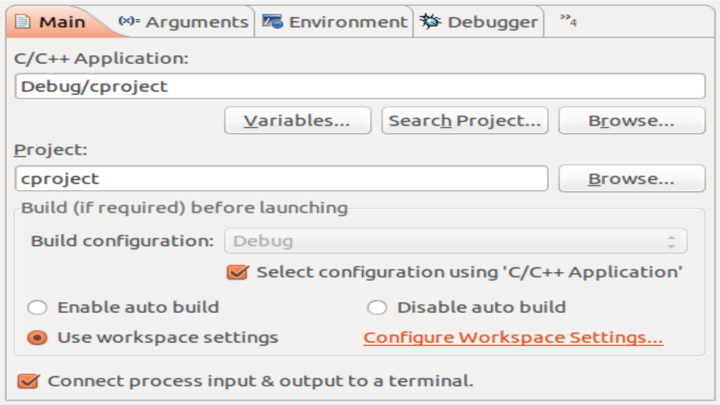
☑ Connect process input & output to a terminal.

Figure 20: Run Configurations

Note that an entry is created under TimeStorm C/C++ remote category with the project name appended with the Active build configuration, and other values are filled in with the default values that should work. The run configuration entries are categories in different tabs on the right hand panel and are explained below. To run the application, mostly you have to just select the remote target on the target tab and click the Run button.

Main Tab

In the Main tab *(shown in Figure 21)*, you can change the C/C++application by searching the project. You can change the project too if you want to use this configuration for a different project.


C/C++ Application: Debug/cproject

Project: cproject

Build (if required) before launching

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

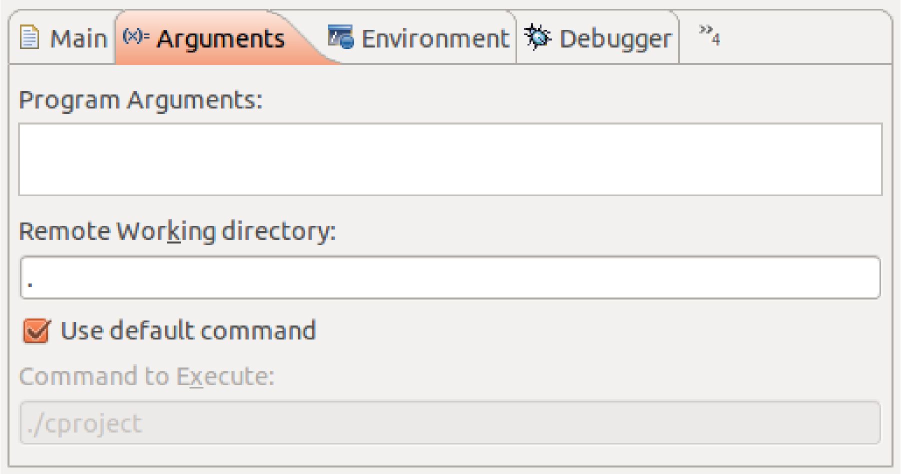
● Use workspace settings Configure Workspace Settings...

☑ Connect process input & output to a terminal.

Figure 21: Main tab

Arguments tab

In the Arguments tab *(shown in figure 22)*, you can specify the arguments to be passed to the application program. The remote working directory is set as '.', which means that the directory specified in the target definition is used as the working directory.



Program Arguments:

Remote Working directory: .

☑ Use default command

Command to Execute: ./cproject

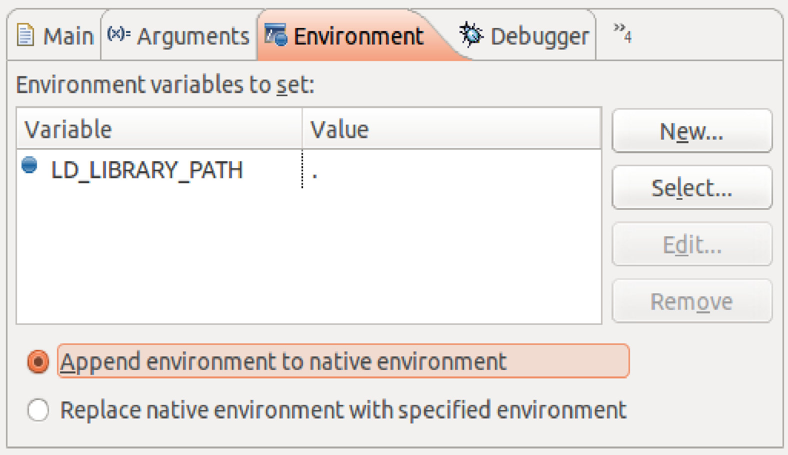
Figure 22: Arguments tab

The command to be executed is filled in. Uncheck 'Use default command' to change this value. Keep in mind the directory structure on your target when constructing this command. For example, if you have copied your application file to a directory named `/apps` but you want to execute it from a working directory named `/home/test`, you must include the path to your application in your command.

For example: `../../apps/my_app`

Environment tab

In the 'Environment' tab *(shown in figure 23)*, you can set environment variables for the target.



Environment variables to set:

Variable	Value
LD_LIBRARY_PATH	.

● Append environment to native environment

○ Replace native environment with specified environment

Figure 23: Environment tab

The panel shown in *figure 23* contains the following buttons:

New — Click the ‘New’ button to create a new entry, and the ‘New Environment Variable’ dialog will appears, as shown in *figure 24* below. Click the ‘Variables’ button to select the variables that you want to use. Then, click ‘OK’ twice.

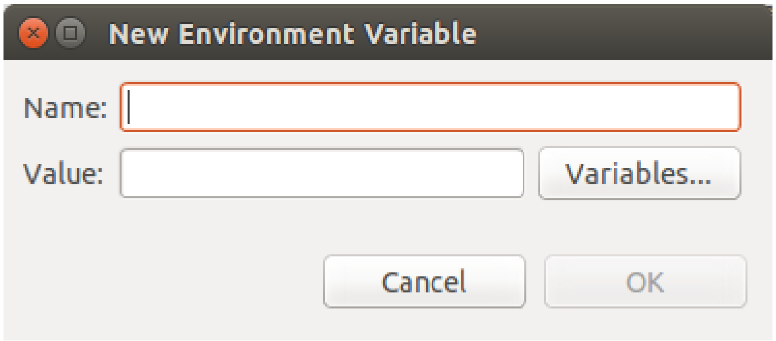


Figure 24: Adding a New Environment Variable

Select — Click the ‘Select‘ button to import environment variables from the host file system.

Edit — To change an entry, select it from the list, and click the ‘Edit’ button.

Remove — To delete an entry, select it from the list, and click the ‘Remove’ button.

Source tab

The Source tab *(figure 25)* shows the location of the source files for the project.

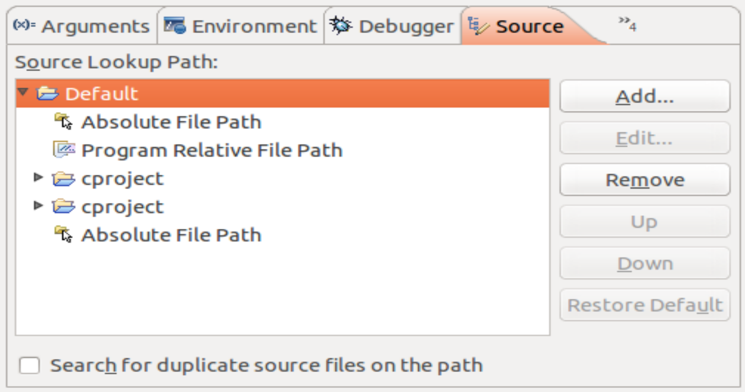


Figure 25: Source tab

Source Lookup Path — This field shows the project being run, as well as any projects that it references.

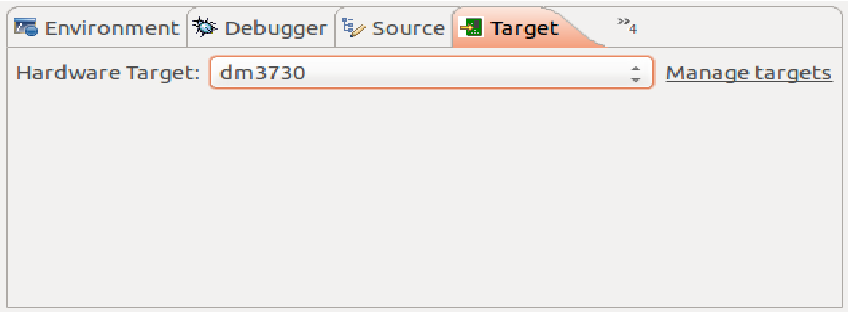
Add — You can add arbitrary source file locations by using the Add button. These locations are searched after the generic locations, from the top to the bottom item in order.

Edit, Remove, Up, Down — The other buttons to the right of the list allow you to edit, remove, or reorder the list items, and to restore the default information.

Search for duplicate source files on the path — Selecting this checkbox causes TimeStorm to notify the user if it is unable to determine which source file corresponds to an executing binary file. For example, if you have two source files with the same filename in different directories of the search path, TimeStorm is unable to determine which file is related to the binary file with that name. If this checkbox is selected, TimeStorm displays a message and asks the user to select which file to use. This checkbox is not selected by default.

Target tab

The Target tab *(shown in figure 26)*, allows you to select the hardware target used when you run the application.



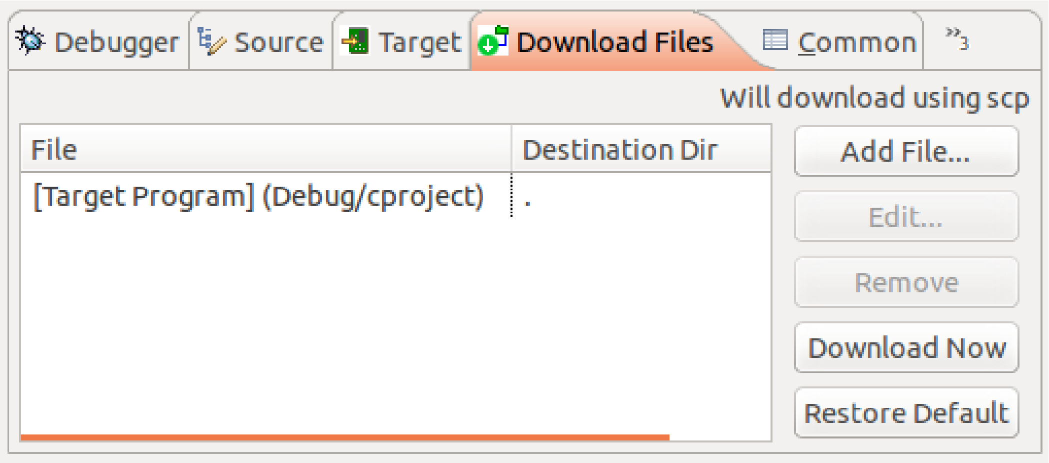
Hardware Target: dm3730

Figure 26: Target tab

In the Target tab, use the drop-down list to select a hardware target that you have already created. If you have not yet created a hardware target or want to change the settings for an existing target, click ‘Manage targets’ to open the Hardware Targets management utility.

Download Files tab

In the ‘Download Files’ tab (*shown in figure 27)*, you can select the files that TimeStorm transfers to your target. The application that has to be downloaded is already listed. You can use this panel to download additional files to your target, along with your application.



Will download using scp

File	Destination Dir
[Target Program] (Debug/cproject)	.

Figure 27: Download Files tab

The application specified in the File field is listed as `[Target Program]`. This is the application that will be run. You can change where it is downloaded by selecting it and clicking Edit.

If the program links in shared libraries, then these libraries will be downloaded to the target as well and the Files field is listed as `[Target Program and Libraries]`. These libraries will be downloaded to the same location as the target program. To download the program and libraries to separate locations, remove the default entry and re-add each as a separate entry.

File paths are interpreted as relative to the target directory to which you download files. If you transfer files by using FTP, the actual destination directories depend on the configuration of your FTP software. In some cases, directories specified in this panel are appended to the user's home directory on the target.

This panel contains the following buttons:

Add File – Use the 'Add File' button to specify additional files to transfer, along with the application. These files are transferred every time you transfer the application file.

Edit – To change where the application (or any other file in this list) is installed, select it and click 'Edit' to change its destination directory.

Remove – To eliminate a file from the items to download, select it in the list and click the 'Remove' button.

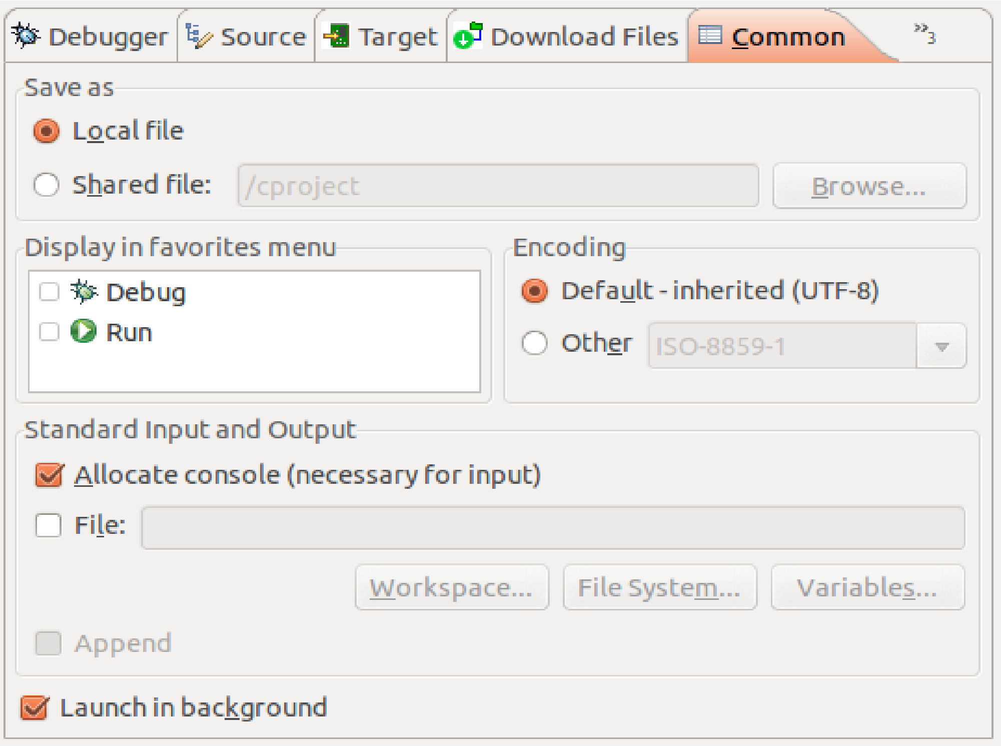
Download Now – The 'Download Now' button copies files immediately, without running the application.

Restore Default – The 'Restore Default' button resets the panel to its initial state (download only the application and its libraries).

> ***NOTE:*** *Downloaded files overwrite any identically named files on the target without giving a warning. TimeStorm will not create destination directories on the target, so be sure to create them before launching the configuration.*

Common tab

The 'Common' tab *(shown in figure 28)*, sets options for sharing this run configuration among multiple projects.



Save as

● Local file

○ Shared file: /cproject   Browse...

Display in favorites menu

□ Debug

□ Run

Encoding

● Default - inherited (UTF-8)

○ Other ISO-8859-1

Standard Input and Output

☑ Allocate console (necessary for input)

□ File:

Workspace...   File System...   Variables...

□ Append

☑ Launch in background

Figure 28: Common tab

The options with the Common tab include:

Local File — By default, run configurations are saved with the workspace state files, so they can only be used with projects in the current workspace. (This setting corresponds to the 'Local file' radio button on this panel.) However, you can make the configuration available for use in other workspaces by choosing the 'Shared file' radio button.

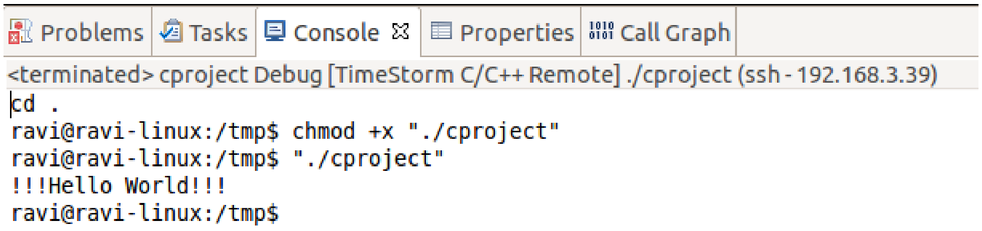
Shared File — When you select this option, the run configuration is saved as a `.launch` file that can be imported into another TimeStorm workspace. Optionally, you can specify a different location for the file so that it is more easily accessible to multiple projects.

Display in Favorites Menu — You can select whether to include this run configuration on the main 'Run' and 'Debug' menus. For example, if you select the 'Run' checkbox, the run configuration always appears in the 'Run History' submenu.

You can choose to include the run configuration in either menu, both menus, or neither menu.

Launch in Background — Selecting the 'Launch in background' checkbox causes this run configuration to run as a background thread. Running as a background thread is the default value. Clear this checkbox if you want to run in the foreground, along with other TimeStorm processing.

After setting the values in all the tabs, click the Run button at the bottom of the run configuration dialog. This will connect your host to your target, download the application to the target and execute the application on the target. The commands executed on the target and the program output progress are displayed in the console view as shown in *figure 29.*



<terminated> cproject Debug [TimeStorm C/C++ Remote] ./cproject (ssh - 192.168.3.39)

```
cd .
ravi@ravi-linux:/tmp$ chmod +x "./cproject"
ravi@ravi-linux:/tmp$ "./cproject"
!!!Hello World!!!
ravi@ravi-linux:/tmp$
```

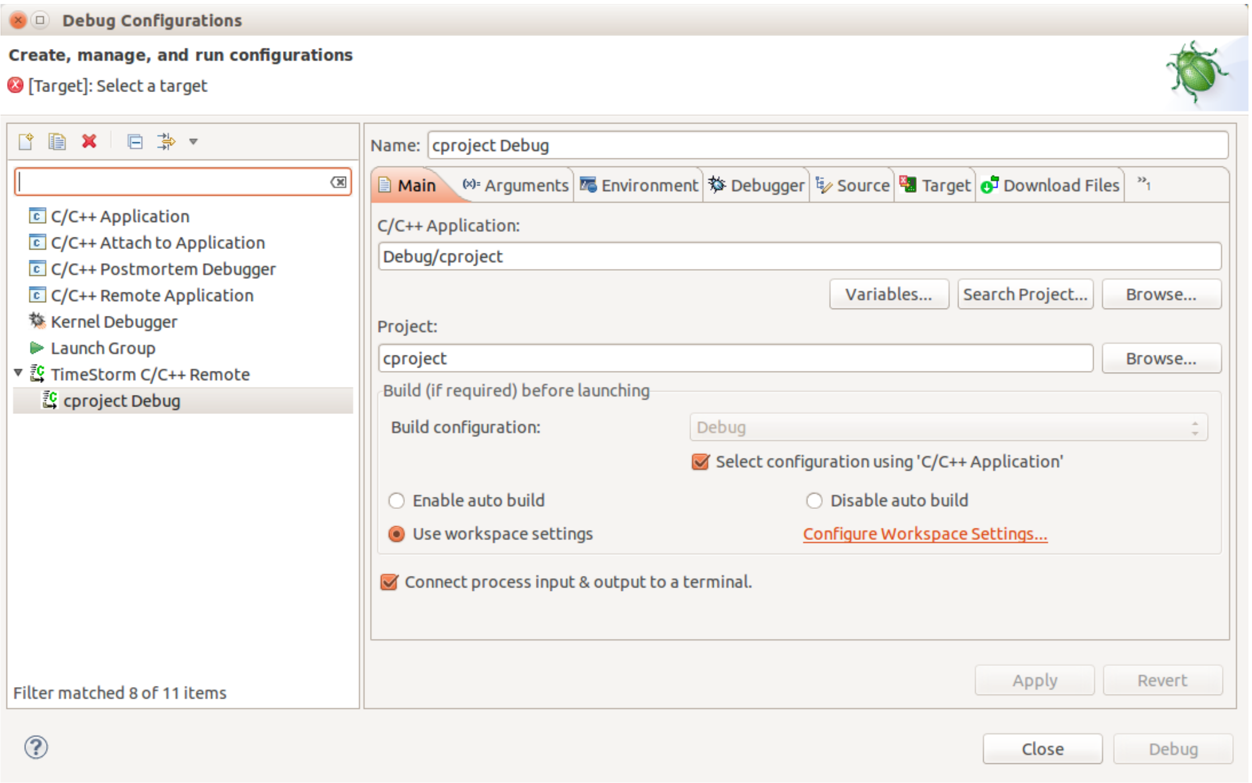
Figure 29: Output of an application run on the target

Debug

Debug Configuration

To debug an application, build the project using the debug build configuration that has the toolchain settings to include complete debug information in the binary. After the project is built, you have to create a debug configuration to debug the application. A debug configuration is similar to a run configuration created in the previous section, but with additional settings for the launching and using the debugger. To create a debug configuration, right click on the project and click on Debug As > Remote C/C++ application in the project context menu. This will open a Debug configurations dialog *as* shown in *figure 30*.

The debug configuration dialog is similar to the run configuration dialog as shown in *figure 20* with an additional debugger tab that includes the settings for the debugger. Most of the values in all the tabs are filled in by default. In most cases, you have to select the target you have to debug on the debug tab and click the debug button to start debugging. By default the debugger will stop at the main function and you can continue debugging from there.



Debug Configurations

Create, manage, and run configurations

[Target]: Select a target

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Kernel Debugger
- Launch Group
- TimeStorm C/C++ Remote
  - cproject Debug

Filter matched 8 of 11 items

Name: cproject Debug

C/C++ Application: Debug/cproject

Project: cproject

Build (if required) before launching

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

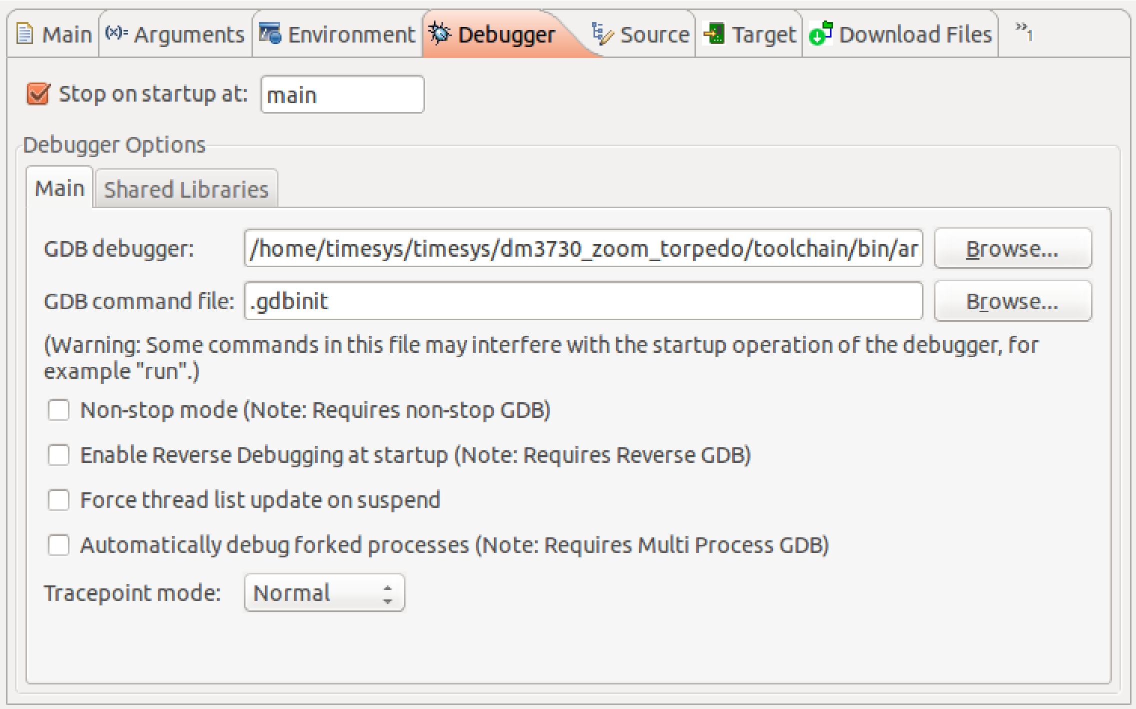
○ Enable auto build ○ Disable auto build

● Use workspace settings Configure Workspace Settings...

☑ Connect process input & output to a terminal.

Figure 30: Debug Configurations

To change the debugger settings, click on the Debugger tab as shown in *figure 31*.



☑ Stop on startup at: main

Debugger Options

GDB debugger: /home/timesys/timesys/dm3730_zoom_torpedo/toolchain/bin/ar

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

□ Non-stop mode (Note: Requires non-stop GDB)

□ Enable Reverse Debugging at startup (Note: Requires Reverse GDB)

□ Force thread list update on suspend

□ Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Figure 31: Debugger tab

This tab contains the following options:

Stop on startup at — Select this checkbox if you want execution to pause when your application starts. (This option is selected by default.)

Specify the function name at which the execution should pause.

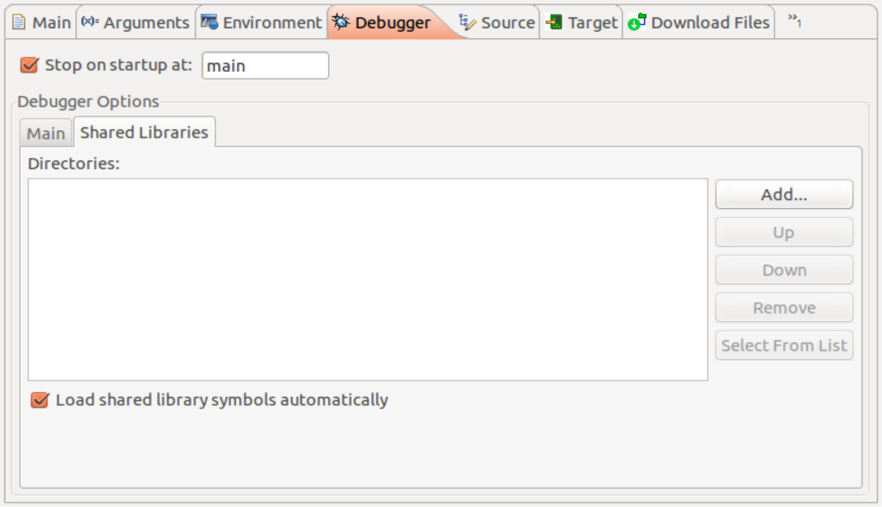
Debugger Options — This section of the panel includes two tabs: 1.) Main and 2.) Shared Libraries. The 'Main' tab is shown in *Figure 16*, above.

Main — This tab includes the following fields:

GDB Debugger — This field is filled in with the gdb from the SDK used with the Active Build Configuration to build the project. Set 'Debug' as the Active Build Configuration to be sure that the correct gdb is filled in.

GDB command file — This field allows you to specify a `.gdb-command` file using the Browse button. The debugger will execute the commands specified in this file.

Shared Libraries — The Shared Libraries tab shown in *figure 32* includes the following options:



☑ Stop on startup at: main

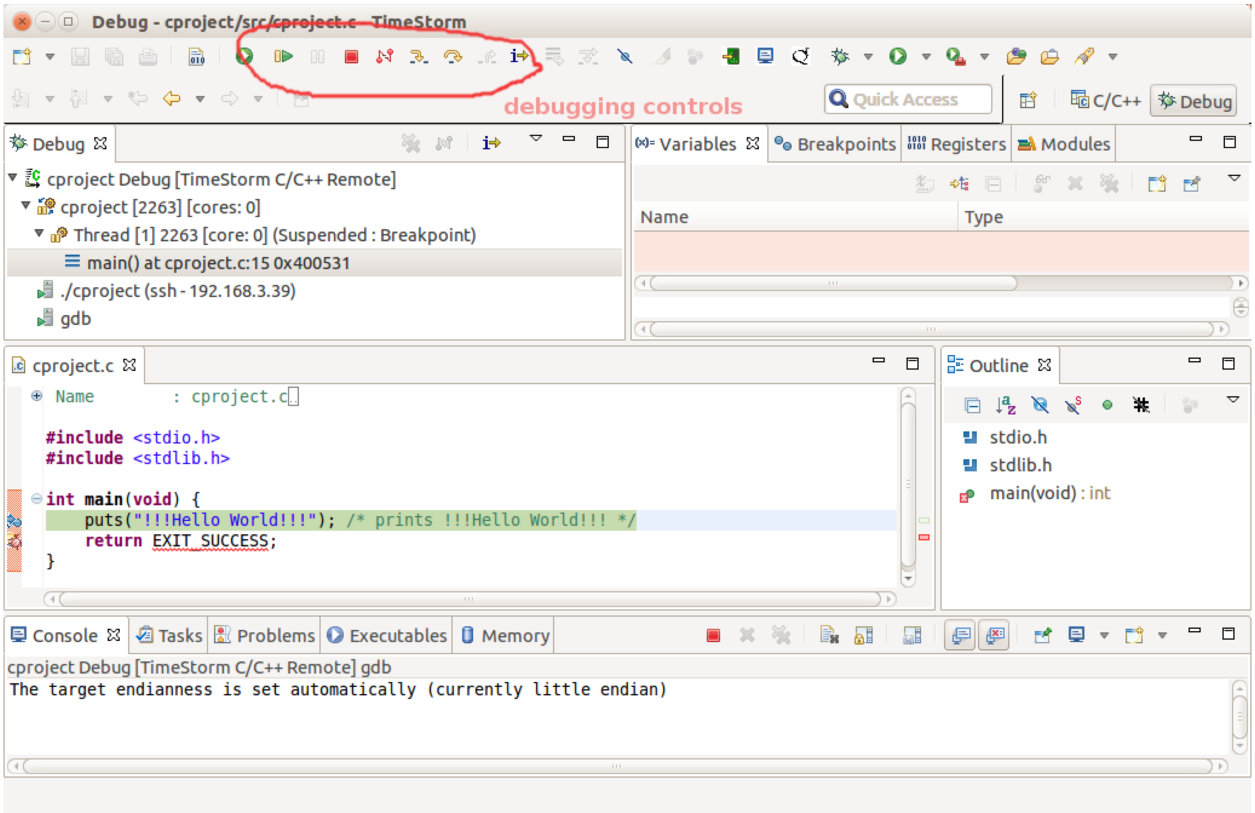
☑ Load shared library symbols automatically

Figure 32: Shared Libraries

Directories — This field specifies any additional directories for the debugger to search to find shared libraries (with debugging symbols). By default the debugger will automatically search the paths specified in the project's build settings (see option below). Use the 'Add' button if you want to add other paths. Use the 'Up' and 'Down' buttons to move through the list of directories. Use the 'Remove' button to delete a path.

Load shared library symbols automatically — Select this checkbox if you want these library symbols to be displayed as loaded in the ‘Shared Libraries’ view and if you want the debugger to hit any breakpoints in the shared library project. (This option is selected by default.)

After modifying the debugger settings, click on the Debug button to start debugging. TimeStorm will connect to the target, download the application, start the gdbserver and prompt you to switch to Debug perspective. Debug perspective is a layout of various views like stack, variables and breakpoints management that help you in debugging. Select the ‘Remember my decision’ checkbox and click on the yes button. The debug perspective will open as shown in *figure 33* and the debugger waits at the breakpoint you have set in the code or the ‘stop at function’ you entered in the debugger tab.



```
cproject Debug [TimeStorm C/C++ Remote]
  cproject [2263] [cores: 0]
    Thread [1] 2263 [core: 0] (Suspended : Breakpoint)
      main() at cproject.c:15 0x400531
  ./cproject (ssh - 192.168.3.39)
  gdb
```

```
 Name        : cproject.c

#include <stdio.h>
#include <stdlib.h>

int main(void) {
	puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
	return EXIT_SUCCESS;
}
```

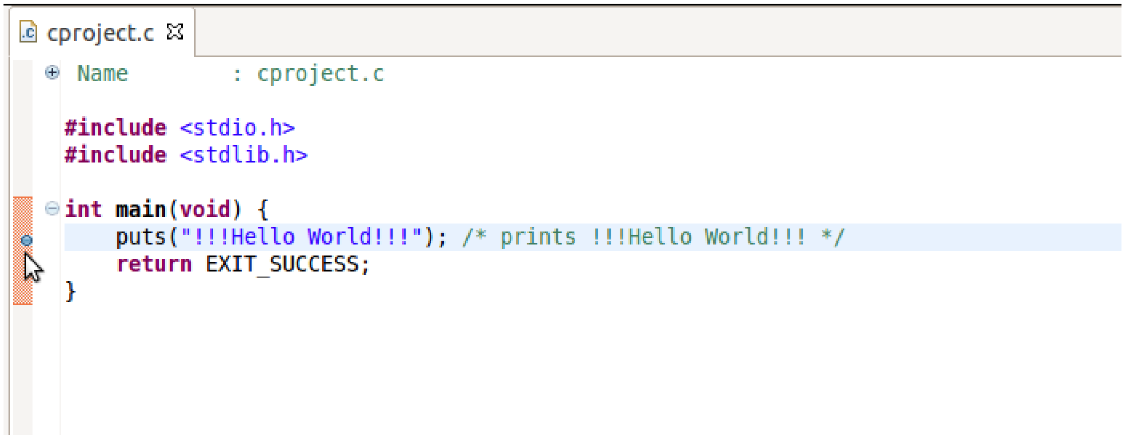
```
cproject Debug [TimeStorm C/C++ Remote] gdb
The target endianness is set automatically (currently little endian)
```

Figure 33: Debug perspective

You can use the buttons circled in the figure above to step in, step out, pause and terminate the debug session. For more details, please refer to C/C++ Development User Guide in TimeStorm help.

Adding a breakpoint

You can add a breakpoint at a line in your source code, by opening the file in the editor and double clicking in the vertical grey bar in the source editor next to the line where you would like to add the breakpoint. This is shown in *figure 34.*



```
 Name        : cproject.c

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
    return EXIT_SUCCESS;
}
```

Figure 34: Adding a breakpoint

Profile

You can profile your application’s code coverage, memory and timing by using many tools included in TimeStorm:

- LTTng - Linux Trace Toolkit, is a high-performance tracing tool for Linux that efficiently handles large amounts of trace data. Initially focused on the Linux kernel, its technology has been extended to support user space tracing (UST). From TimeStorm, you can configure and control LTTng, collect the trace data, and visualize and analyze the trace data.

- OProfile - A powerful profiling tool and using TimeStorm you will be able to configure OProfile, gather OProfile data and visualize it. In addition to the OProfile tool from Eclipse, TimeStorm includes a “Linux Profiling Suite” for easily profiling applications using OProfile. Linux Profiling Suite is explained in the next section.

- Valgrind - A powerful tool that can be used from TimeStorm for profiling applications. TimeStorm adds support to use valgrind on a remote target.

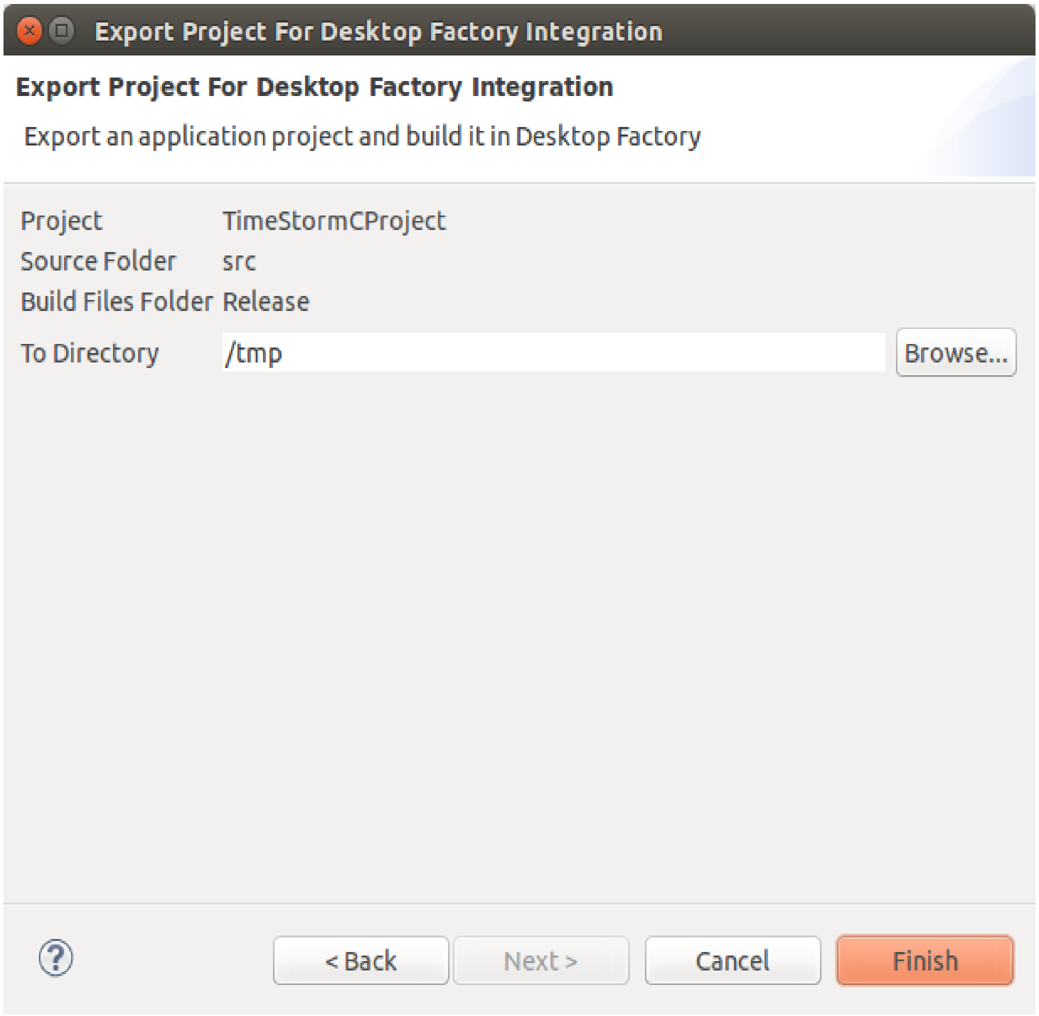
- Gcov - Test code coverage in program using gcov and visualize the gcda and gcno files generated by gcov intrumentation. To learn how to use gcov with TimeStorm, read https://linuxlink.timesys.com/docs/wiki/engineering/HOWTO_Use_Gcov_with_TimeStorm

- GProf - Profile an application using gprof and visualize the gmon.out files generated by gprof instrumentation. To learn how to use gprof with TimeStorm, read https://linuxlink.timesys.com/docs/wiki/engineering/HOWTO_Use_Gprof_with_TimeStorm

Integrate

After you have developed your application in TimeStorm, you can integrate the project into desktop factory that is used by your company’s build system. The project will be built using the toolchain that is used by desktop factory.

To integrate an application into desktop factory, build the project in TimeStorm, right click on the project and click Export > C/C++ > Desktop Factory Integration and click Next. This will open a desktop factory integration dialog as shown below in *figure 35.*



Export Project For Desktop Factory Integration

Export Project For Desktop Factory Integration

Export an application project and build it in Desktop Factory

Project TimeStormCProject
Source Folder src
Build Files Folder Release
To Directory /tmp

Figure 35: Desktop Factory Integration

Select the directory to copy the exported project tar file and click Finish.

You can share the exported tar file with your build engineer / platform developer to integrate into Desktop Factory. They have to run

```
./bin/install_timestorm_project <project.tar.gz>
```

to integrate the project to desktop factory. For qt projects, they have to append `qt` to the above command. For more details, refer to the desktop factory documentation.

Share

To share your projects with other developers, build and test systems, and to manage your project code it is recommended to include your project in a version control system. You can use any version control system of your choice like Perforce, Subversion, Clearcase etc. Eclipse plugins are available for most of the version control systems. TimeStorm includes EGit plugins that simplify using Git as your version control system. Refer to EGit Documentation in TimeStorm help for more details.

Kernel Development with TimeStorm

Kernel Development includes:

- creating a kernel project
- configuring the kernel
- building and deploying the kernel
- debugging the kernel

Creating a Kernel Project

New Project Wizard

To create a kernel project, open the New Project wizard by clicking File > New > Project from main menu or Click the ‘New’ button . Select Kernel > Linux Kernel Project and click ‘Next’ to continue, as shown in *figure 36.*

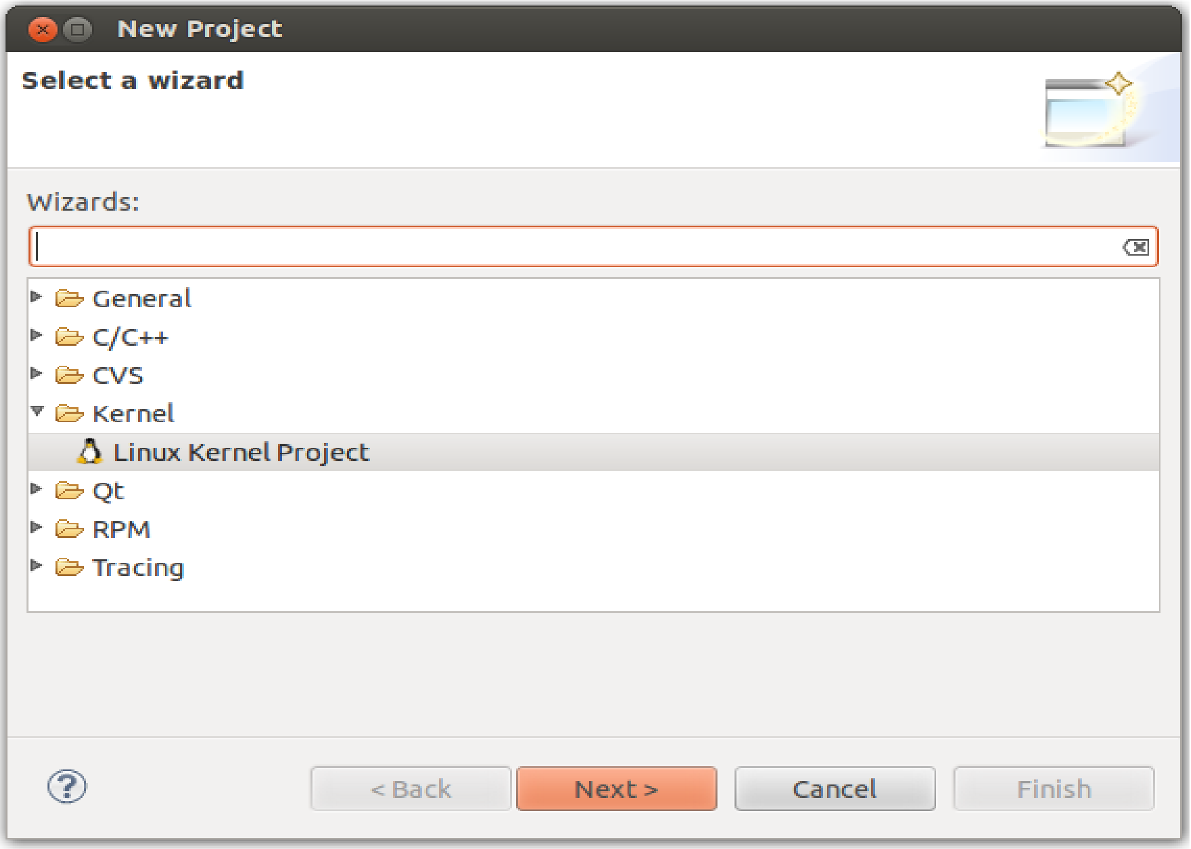
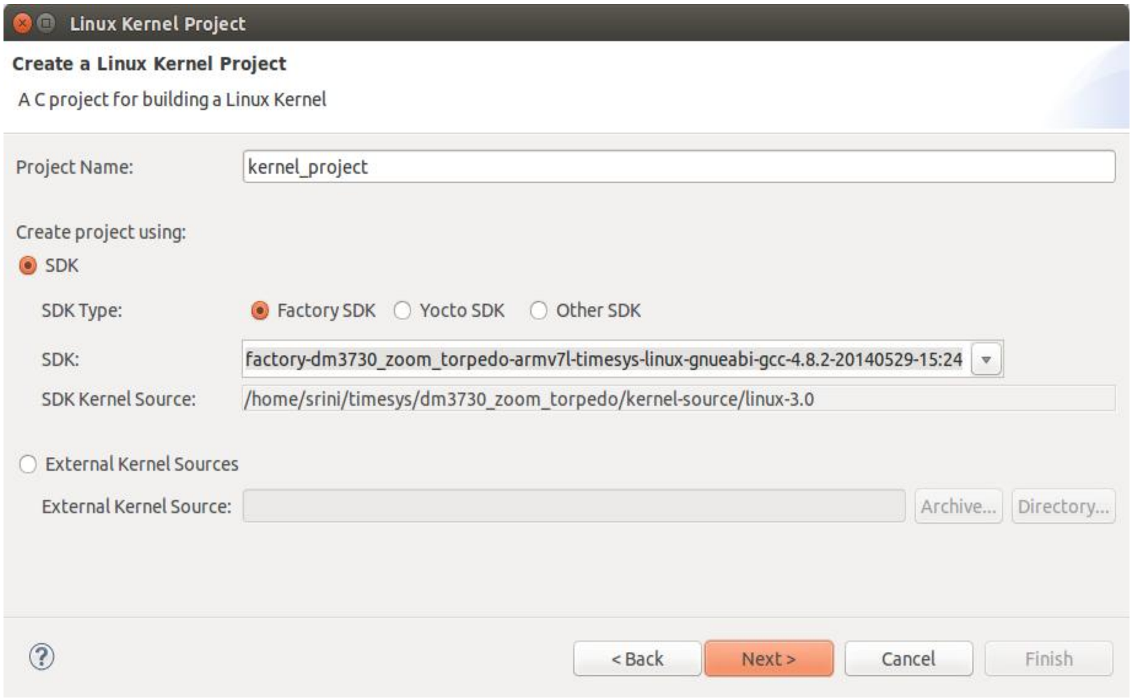


Figure 36: New Project wizard – TimeStorm Linux Kernel Project

Project Page

The page shown in *figure 37* allows you to enter a name for the kernel project, and select the kernel sources to use with the project.



Linux Kernel Project

Create a Linux Kernel Project

A C project for building a Linux Kernel

Project Name: kernel_project

Create project using:

● SDK

SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

SDK: factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24

SDK Kernel Source: /home/srini/timesys/dm3730_zoom_torpedo/kernel-source/linux-3.0

○ External Kernel Sources

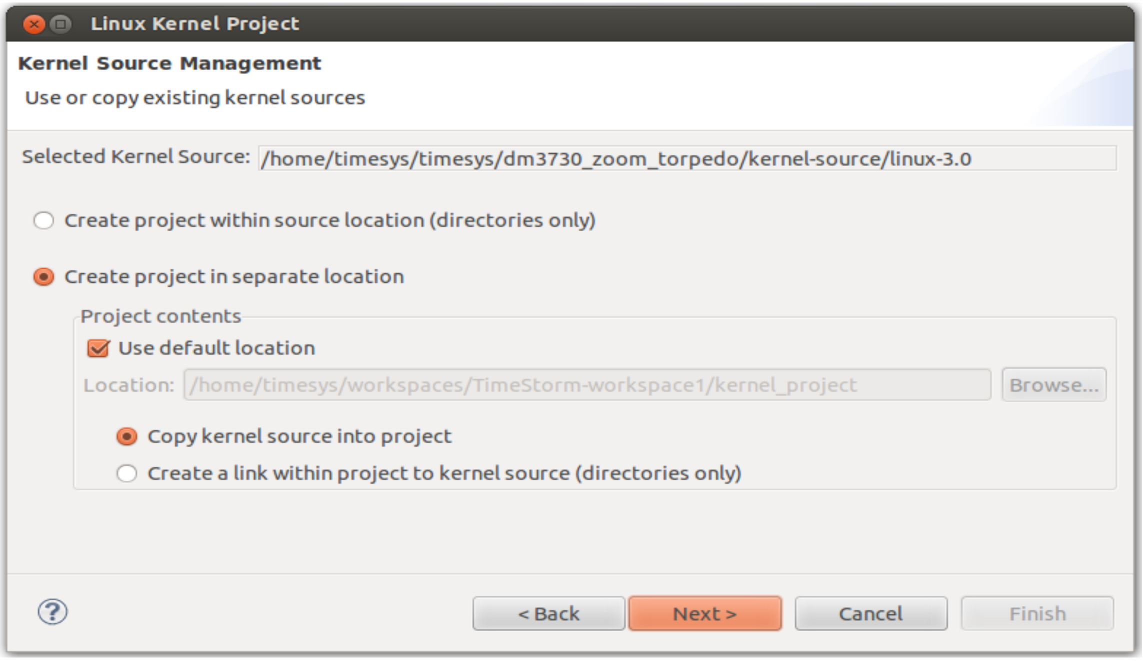
External Kernel Source:

Figure 37: Kernel Project Page

You can create a kernel project using the kernel sources included in the SDK (Factory/Yocto/Other), or kernel sources you have in an archive file or directory. When you select the SDK, the architecture, toolchain and other details required for building the kernel project are filled in for you on the next pages.

Kernel Source Management

You can create a kernel project by making a copy of your existing kernel sources or use your kernel sources. This page allows you to choose how you want to manage your kernel sources. You can create the project in your kernel source location or at a different location. You can also choose to create the project in TimeStorm workspace or at a different location as shown in *figure 38*.



Linux Kernel Project

Kernel Source Management

Use or copy existing kernel sources

Selected Kernel Source: /home/timesys/timesys/dm3730_zoom_torpedo/kernel-source/linux-3.0

○ Create project within source location (directories only)

● Create project in separate location

Project contents

☑ Use default location

Location: /home/timesys/workspaces/TimeStorm-workspace1/kernel_project

● Copy kernel source into project

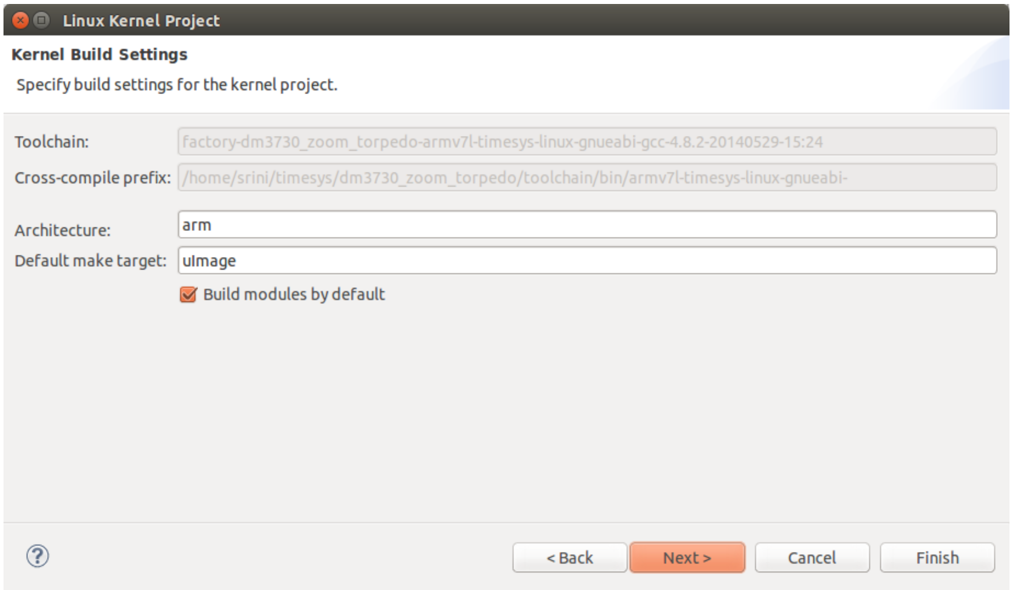
○ Create a link within project to kernel source (directories only)

Figure 38: Kernel source management

Kernel Build Settings

Kernel build settings page show in *figure 39* allows you to enter the architecture and make target. If the SDK selected on the previous page has this information then these values are filled in. Architecture is mandatory and 'Default Make Target' is optional. You can choose whether to build the kernel modules along with the kernel.

The cross-compile prefix for the kernel build is set based on the selected SDK.



Linux Kernel Project

Kernel Build Settings

Specify build settings for the kernel project.

Toolchain: factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24

Cross-compile prefix: /home/srini/timesys/dm3730_zoom_torpedo/toolchain/bin/armv7l-timesys-linux-gnueabi-

Architecture: arm

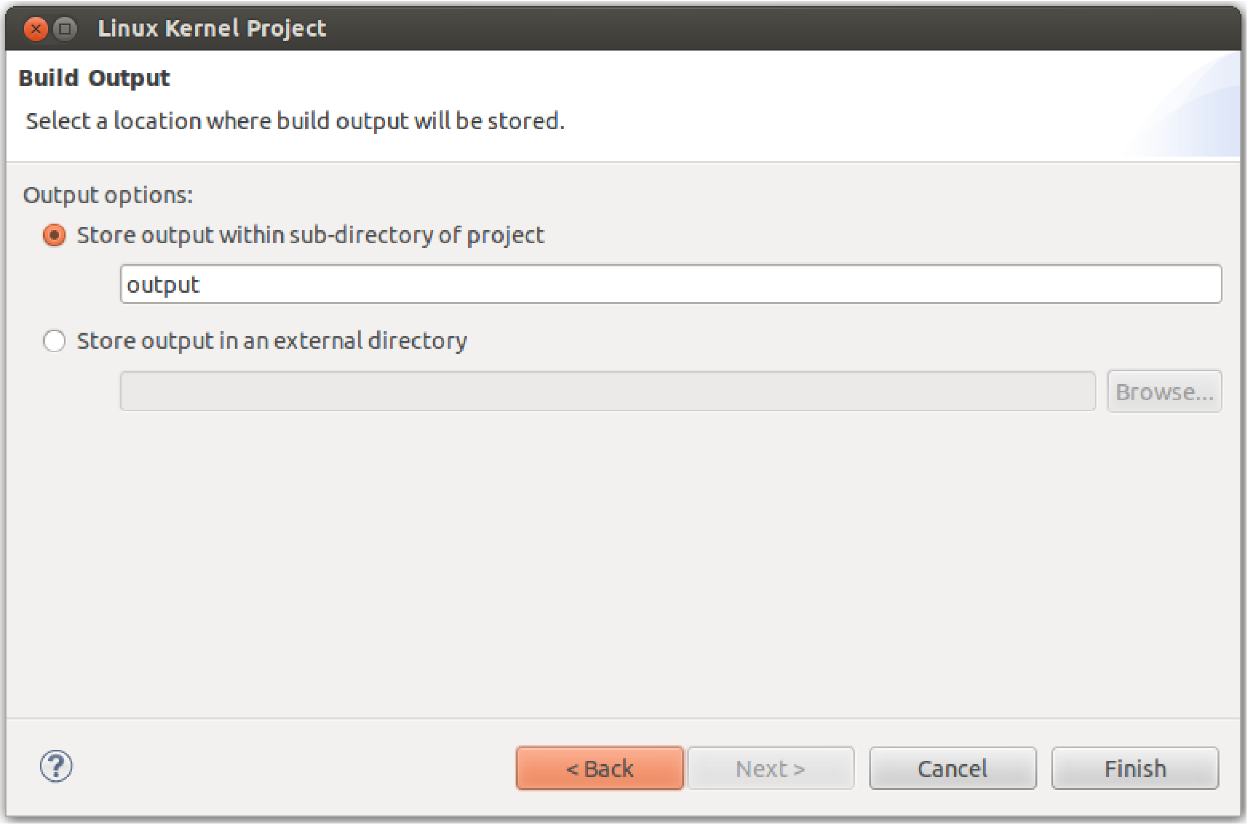
Default make target: uImage

☑ Build modules by default

Figure 39: Kernel Build Settings

Build Output

Figure 40 allows you to manage the kernel build output. You can save the kernel build output in a sub-directory in kernel project or an external directory.



Linux Kernel Project

Build Output

Select a location where build output will be stored.

Output options:

● Store output within sub-directory of project

output

○ Store output in an external directory

Figure 40: Kernel Build Output

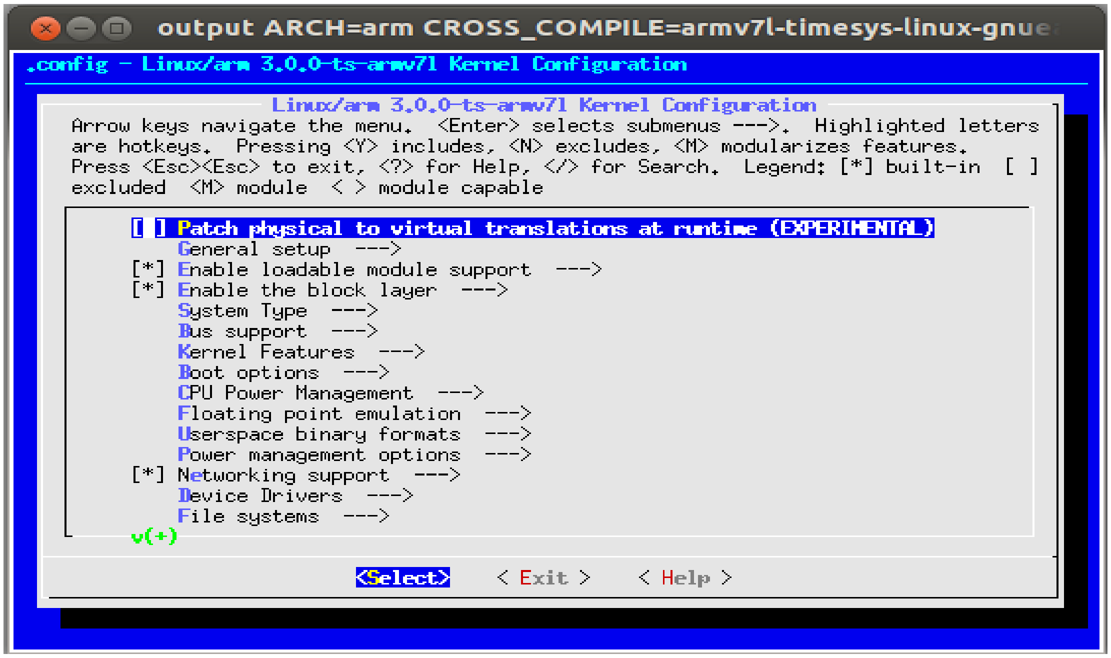
Click Finish to create the kernel project. Copying the kernel sources takes some time and you may notice a delay in creating the project.

Please note that to conserve memory and CPU resources, indexing is turned off for kernel projects. The side effect of this is you may see errors / warnings when you open kernel sources in an editor.

Configuring the Kernel

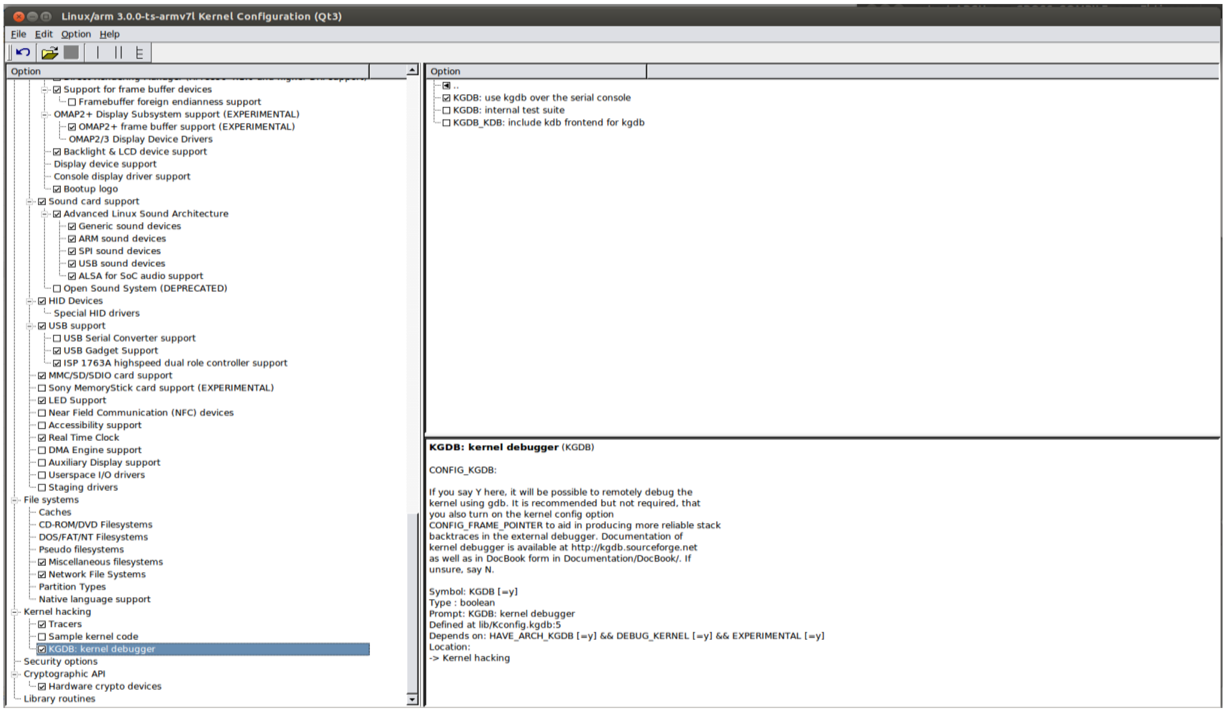
You can configure the kernel by right clicking on the kernel project, clicking Configure Kernel and then clicking MenuConfig or XConfig. TimeStorm launches the menuconfig or xconfig editor outside TimeStorm. You can configure the kernel and save your changes. Please note that for running menuconfig, you should have libncurses5-dev and xterm package installed, and for running xconfig, you should have qt3-dev-tools or qt4-dev-tools package and xterm installed.

Menuconfig and xconfig editors are shown in *figure 41*and *figure 42* respectively.



```
output ARCH=arm CROSS_COMPILE=armv7l-timesys-linux-gnue
.config - Linux/arm 3.0.0-ts-armv7l Kernel Configuration
                    Linux/arm 3.0.0-ts-armv7l Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters
  are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

       [ ] Patch physical to virtual translations at runtime (EXPERIMENTAL)
           General setup  --->
       [*] Enable loadable module support  --->
       [*] Enable the block layer  --->
           System Type  --->
           Bus support  --->
           Kernel Features  --->
           Boot options  --->
           CPU Power Management  --->
           Floating point emulation  --->
           Userspace binary formats  --->
           Power management options  --->
       [*] Networking support  --->
           Device Drivers  --->
           File systems  --->
       v(+)

                      <Select>    < Exit >    < Help >
```

Figure 41: Menuconfig – Kernel Configuration



```
  ☑ Support for frame buffer devices
    □ Framebuffer foreign endianness support
  OMAP2+ Display Subsystem support (EXPERIMENTAL)
    ☑ OMAP2+ frame buffer support (EXPERIMENTAL)
    OMAP2/3 Display Device Drivers
  ☑ Backlight & LCD device support
  Display device support
  Console display driver support
  ☑ Bootup logo
☑ Sound card support
  ☑ Advanced Linux Sound Architecture
    ☑ Generic sound devices
    ☑ ARM sound devices
    ☑ SPI sound devices
    ☑ USB sound devices
    ☑ ALSA for SoC audio support
  □ Open Sound System (DEPRECATED)
☑ HID Devices
  Special HID drivers
☑ USB support
  □ USB Serial Converter support
  ☑ USB Gadget Support
  ☑ ISP 1763A highspeed dual role controller support
☑ MMC/SD/SDIO card support
□ Sony MemoryStick card support (EXPERIMENTAL)
☑ LED Support
□ Near Field Communication (NFC) devices
□ Accessibility support
☑ Real Time Clock
□ DMA Engine support
□ Auxiliary Display support
□ Userspace I/O drivers
□ Staging drivers
File systems
  Caches
  CD-ROM/DVD Filesystems
  DOS/FAT/NT Filesystems
  Pseudo filesystems
  ☑ Miscellaneous filesystems
  ☑ Network File Systems
  Partition Types
  Native language support
Kernel hacking
  ☑ Tracers
  □ Sample kernel code
  ☑ KGDB: kernel debugger
Security options
Cryptographic API
  ☑ Hardware crypto devices
Library routines
```

```
..
☑ KGDB: use kgdb over the serial console
□ KGDB: internal test suite
□ KGDB_KDB: include kdb frontend for kgdb
```

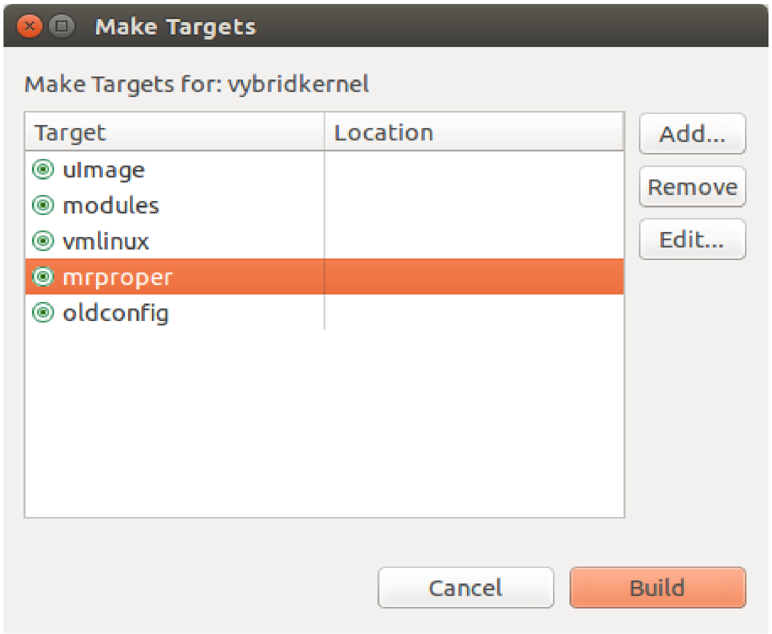
```
KGDB: kernel debugger (KGDB)

CONFIG_KGDB:

If you say Y here, it will be possible to remotely debug the
kernel using gdb. It is recommended but not required, that
you also turn on the kernel config option
CONFIG_FRAME_POINTER to aid in producing more reliable stack
backtraces in the external debugger. Documentation of
kernel debugger is available at http://kgdb.sourceforge.net
as well as in DocBook form in Documentation/DocBook/. If
unsure, say N.

Symbol: KGDB [=y]
Type : boolean
Prompt: KGDB: kernel debugger
Defined at lib/Kconfig.kgdb:5
Depends on: HAVE_ARCH_KGDB [=y] && DEBUG_KERNEL [=y] && EXPERIMENTAL [=y]
Location:
-> Kernel hacking
```

Figure 42: XConfig – Kernel configuration

Building and Deploying the Kernel

To build a kernel project right click on the project and click on build project. The build progress is displayed in the console view.

While building the kernel, sometimes you may see that you have to run mrproper. You can run mrproper by using the make targets included with TimeStorm kernel project. To launch the Make Targets window, right click on the kernel project and click on Make Targets > Build …, and the make targets window will display as shown in *figure 43.*



Make Targets for: vybridkernel

Target	Location
uImage	
modules	
vmlinux	
mrproper	
oldconfig	

Figure 43: Managing make targets

To run a make target, select the target from the list and click Build.

Booting the board with the kernel image that is built is specific to the board. Please refer the getting started guide for your board for instructions on how to deploy the kernel image and boot the board. TimeStorm does include any features for deploying the kernel and booting the board.

Debugging the Kernel

To debug a kernel, you have to:

- configure the kernel for debugging
- build the kernel
- boot the board with the kernel using the right boot arguments
- debug the kernel from TimeStorm using gdb

Configure the kernel for debugging

To configure the kernel for debugging, open menuconfig or xconfig for the kernel project and:

- In “General Setup”, turn on “Prompt for development and/or incomplete code/drivers” (CONFIG_EXPERIMENTAL)
- In “Kernel Hacking”, turn on “Kernel Debugging” , “Compile the kernel with debug info”, and “KGDB: kernel debugger”

- In "Kernel Hacking" > "KGDB: kernel debugger", turn on "KGDB: use kgdb over the serial console".
- For additional info, refer to https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/CompilingAKernel.html#CompileKGDB

Build the kernel

After configuring the kernel for debugging, save the kernel configuration and build the kernel by clicking on 'Build Project' in the project context menu.

Boot the Board with the kernel using the right boot arguments

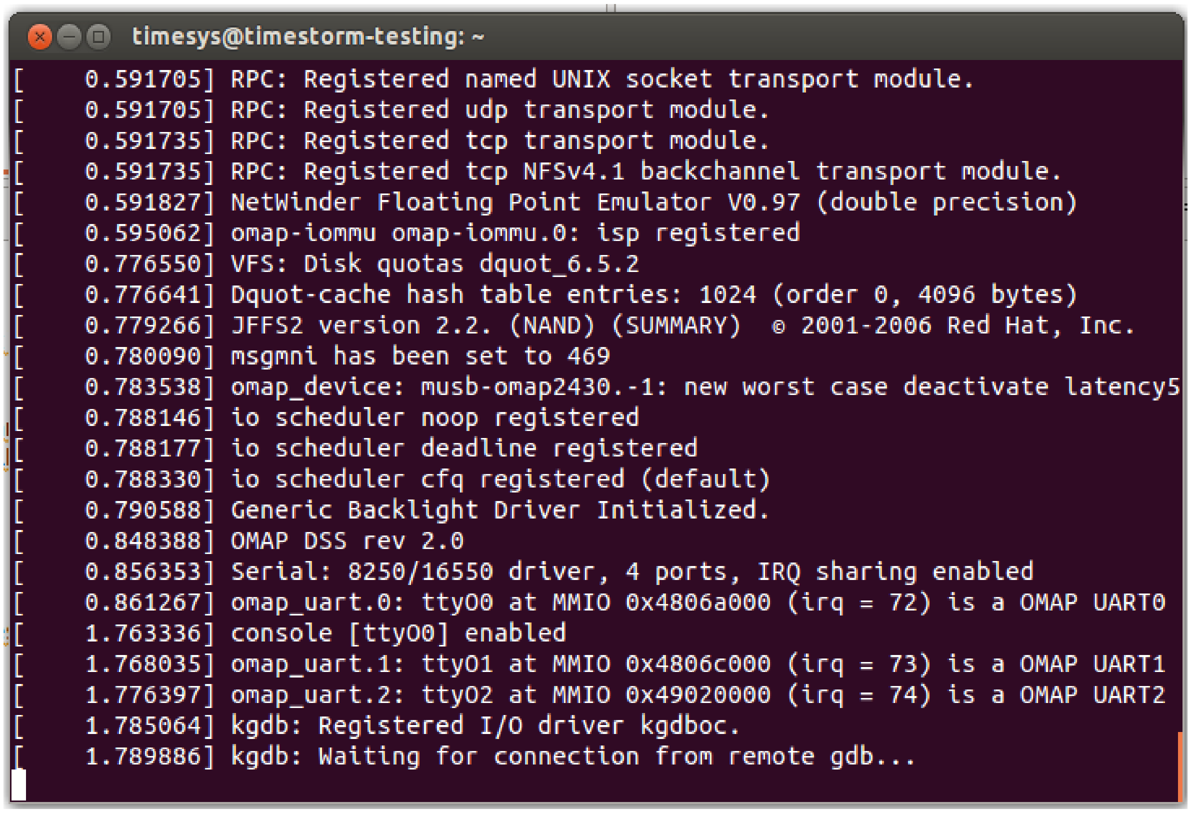
The additional boot arguments passed to the kernel include the gdb and kgdb communication setting, and instructing the kernel to wait for the gdb connection. Gdb communicates to kgdb over serial connection and you set the serial device and the baud rate as the boot arguments.

For kernel debugging append these arguments to your boot arguments.

```
kgdboc=<serial_device>,<baud> kgdbwait
```

For additional information on the boot arguments, refer to https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/kgdbKernelArgs.html

When you boot the kernel with these boot arguments, kgdb registers the kgdboc driver and waits for a remote gdb connection, as shown below in *figure 44.*



```
[    0.591705] RPC: Registered named UNIX socket transport module.
[    0.591705] RPC: Registered udp transport module.
[    0.591735] RPC: Registered tcp transport module.
[    0.591735] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.591827] NetWinder Floating Point Emulator V0.97 (double precision)
[    0.595062] omap-iommu omap-iommu.0: isp registered
[    0.776550] VFS: Disk quotas dquot_6.5.2
[    0.776641] Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
[    0.779266] JFFS2 version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, Inc.
[    0.780090] msgmni has been set to 469
[    0.783538] omap_device: musb-omap2430.-1: new worst case deactivate latency5
[    0.788146] io scheduler noop registered
[    0.788177] io scheduler deadline registered
[    0.788330] io scheduler cfq registered (default)
[    0.790588] Generic Backlight Driver Initialized.
[    0.848388] OMAP DSS rev 2.0
[    0.856353] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
[    0.861267] omap_uart.0: ttyO0 at MMIO 0x4806a000 (irq = 72) is a OMAP UART0
[    1.763336] console [ttyO0] enabled
[    1.768035] omap_uart.1: ttyO1 at MMIO 0x4806c000 (irq = 73) is a OMAP UART1
[    1.776397] omap_uart.2: ttyO2 at MMIO 0x49020000 (irq = 74) is a OMAP UART2
[    1.785064] kgdb: Registered I/O driver kgdboc.
[    1.789886] kgdb: Waiting for connection from remote gdb...
```

Figure 44: Kernel boot console log – kgdb waiting for gdb connection

Debug the kernel from TimeStorm using gdb

To debug a kernel,

1. Set a breakpoint :

    a. Open the kernel source you want to debug in the source editor

    b. Set a break point by either double clicking at the far left end of the line in the source editor or by right clicking at the far left end of the line in the source editor and click “Add Breakpoint”.

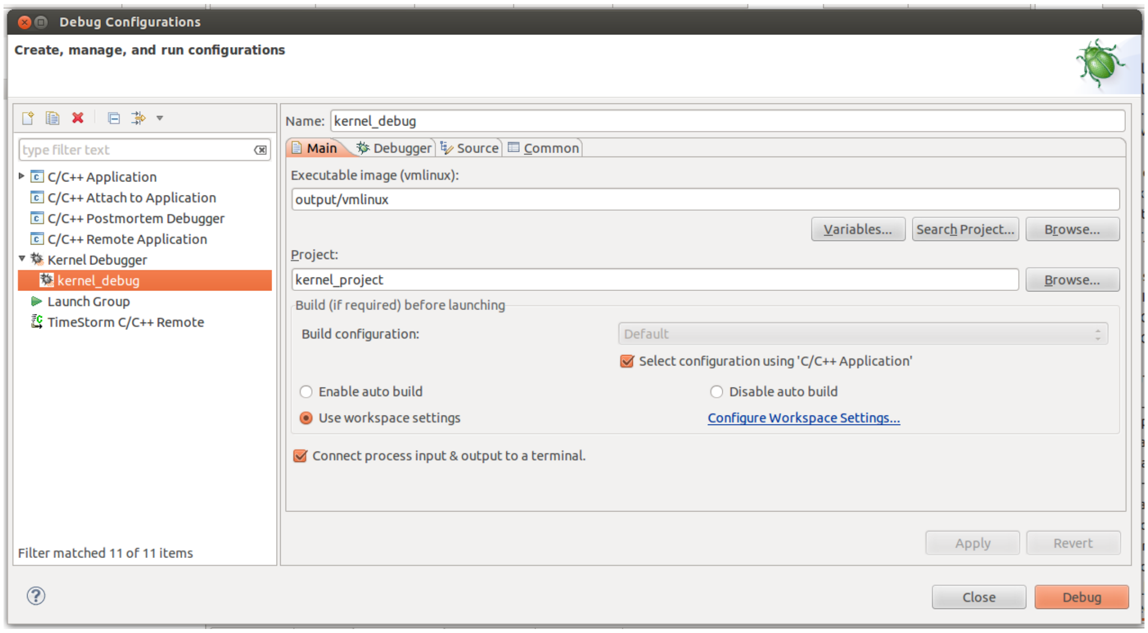
2. Create a debug launch configuration and start debugging.

    a. Click and select the kernel project you want to debug

    b. Right click on the kernel project and click on Debug As > Debug Configurations to open the Debug Configurations dialog.

    c. Select Kernel Debugger in the left panel.

d. Click New launch configuration icon in the top left corner. This will create a new debug launch configuration for the kernel project and the launch configuration tabs are displayed in the right panel as shown in *figure 45*. For a kernel that is already built, most of the entries in all the fields are already filled in and you may not have to change any entries for debugging. You have to set the gdb connection settings in the "Connection tab" in the "Debugger" tab.



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- Kernel Debugger
  - kernel_debug
- Launch Group
- TimeStorm C/C++ Remote

Filter matched 11 of 11 items

Name: kernel_debug

Main | Debugger | Source | Common

Executable image (vmlinux): output/vmlinux

Variables... | Search Project... | Browse...

Project: kernel_project

Browse...

Build (if required) before launching

Build configuration: Default

☑ Select configuration using 'C/C++ Application'

○ Enable auto build   ○ Disable auto build

● Use workspace settings   Configure Workspace Settings...

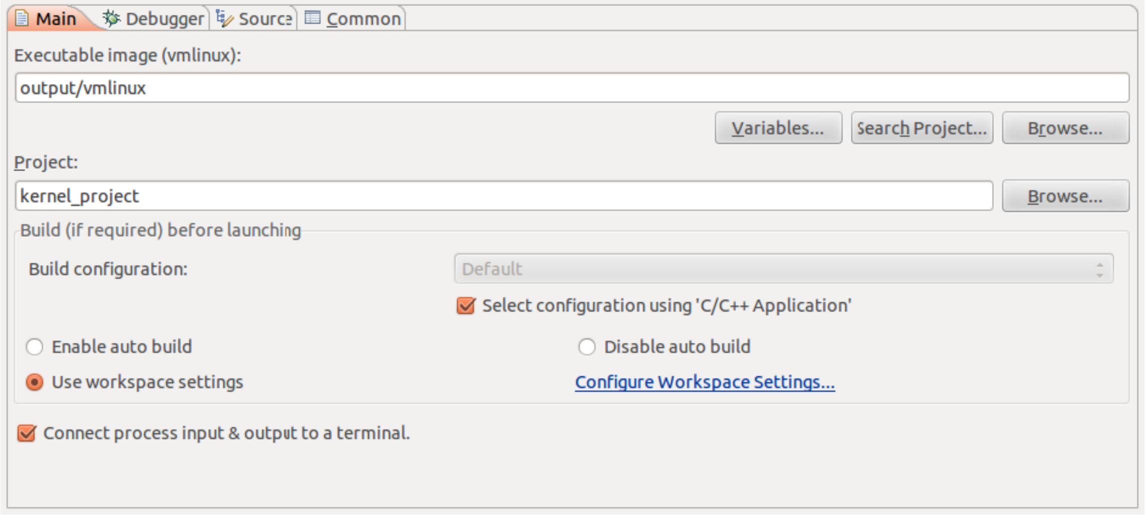
☑ Connect process input & output to a terminal.

Apply | Revert

Close | Debug

Figure 45: Debug launch configuration for kernel project

Main tab: Enter a name for the debug launch configuration. Enter the kernel project to debug by your workspace by clicking the browse button. Select and enter the vmlinux file in your kernel project.



Main | Debugger | Source | Common

Executable image (vmlinux): output/vmlinux

Variables... | Search Project... | Browse...

Project: kernel_project

Browse...

Build (if required) before launching

Build configuration: Default

☑ Select configuration using 'C/C++ Application'

○ Enable auto build   ○ Disable auto build

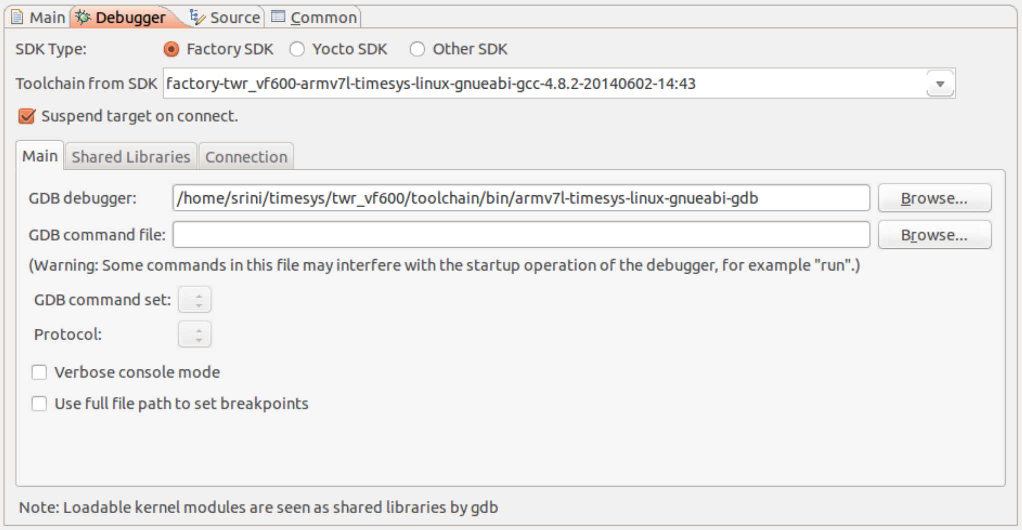
● Use workspace settings   Configure Workspace Settings...

☑ Connect process input & output to a terminal.

Figure 46: Kernel Debug Configuration – Main tab

Debugger Tab: If you want to change the gdb used to debug the kernel, then you can select the SDK Type and choose SDK from the dropdown box at the top of the tab.

Main tab: The gdb in the toolchain is filled in the GDB debugger text field. If you want to change the gdb, you can do so by clicking the Browse button next to the gdb debugger text field. If you wish not to run any commands when gdb launches, you can clear ".init" entry in the gdb command file text field. If you would like to view verbose messages in the gdb console, select the "Verbose console mode" checkbox.



SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK: factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140602-14:43

☑ Suspend target on connect.

GDB debugger: /home/srini/timesys/twr_vf600/toolchain/bin/armv7l-timesys-linux-gnueabi-gdb

GDB command file:

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

GDB command set:

Protocol:

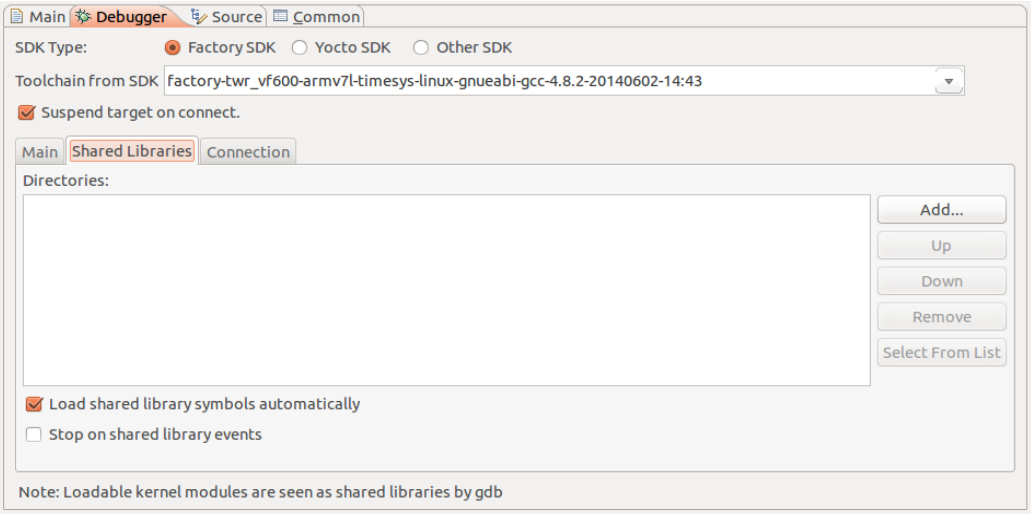
□ Verbose console mode

□ Use full file path to set breakpoints

Note: Loadable kernel modules are seen as shared libraries by gdb

Figure 47: Kernel Debug Configuration: Debugger Tab – Main Tab

Shared Libraries Tab: You can manage the directories the debugger searches to find the shared libraries (with debug symbols) on the "Shared Libraries" tab in the Debugger tab. By default the debugger will automatically search the paths specified in the project's build settings.



SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK: factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140602-14:43

☑ Suspend target on connect.

Directories:

☑ Load shared library symbols automatically

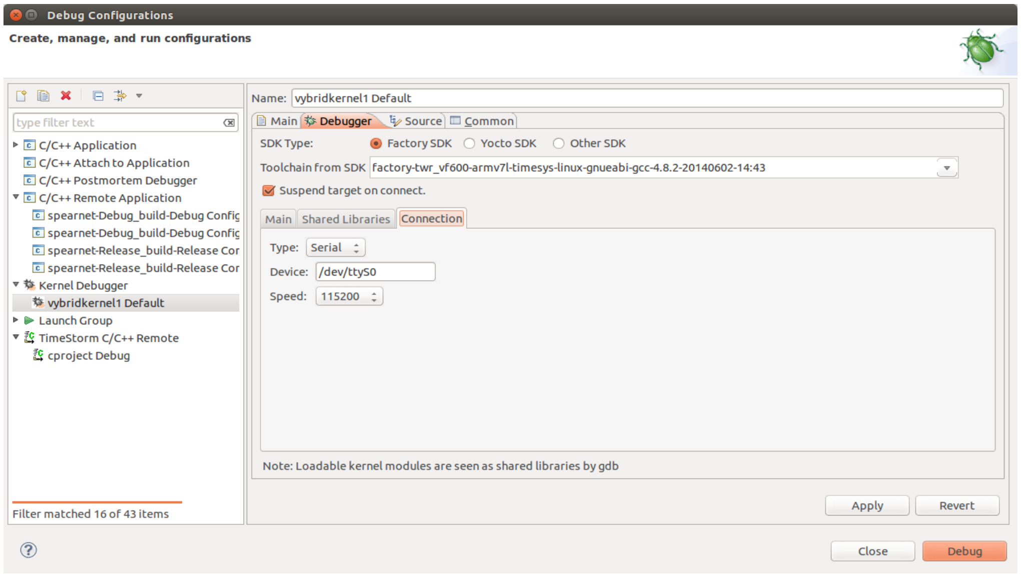
□ Stop on shared library events

Note: Loadable kernel modules are seen as shared libraries by gdb

Figure 48: Kernel Debug Configuration: Debugger Tab –Shared Libraries Tab

“Load shared library symbols automatically” check box is selected by default and this will allow the debugger to hit breakpoints in the shared library. Select the **“**Stop on shared library events” if you want the debugger to stop on shared library events, even before it hits breakpoints in source code.

Connection Tab: kgdb works only over serial port. So select Serial as the connection type, enter the host serial device that is connected to the target board and the baud rate. This baud rate should match the baud entered in the kernel boot argument.



Debug Configurations

Create, manage, and run configurations

Name: vybridkernel1 Default

SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK factory-twr_vf600-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140602-14:43

☑ Suspend target on connect.

Type: Serial

Device: /dev/ttyS0

Speed: 115200

Note: Loadable kernel modules are seen as shared libraries by gdb

Filter matched 16 of 43 items

Figure 49: Kernel Debug Configuration: Debugger Tab – Connection Tab

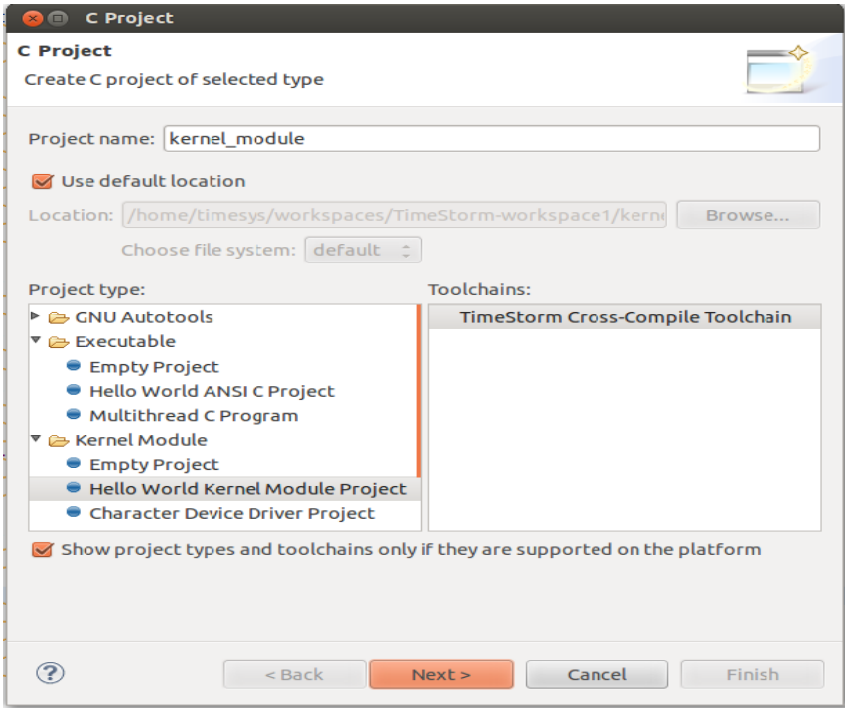
e. Click the debug button to start debugging. TimeStorm will switch to debug perspective and the debugger will stop at the first break point. You will be able to step in, step over and resume through the kernel source code.

Loadable Kernel Module

A loadable kernel module is a special type of C/C++ project used to write device drivers. The loadable kernel module is developed referring specific kernel sources and kernel image and the board should be booted with the same kernel image.

Creating a Lodable Kernel Module Project

Top open a new loadable kernel module project click File > New > Project > C/C++, click Next and choose Kernel Module as the project type as shown in *figure 50.*



C Project

Create C project of selected type

Project name: kernel_module

☑ Use default location

Location: /home/timesys/workspaces/TimeStorm-workspace1/kern

Choose file system: default

Project type: Hello World Kernel Module Project

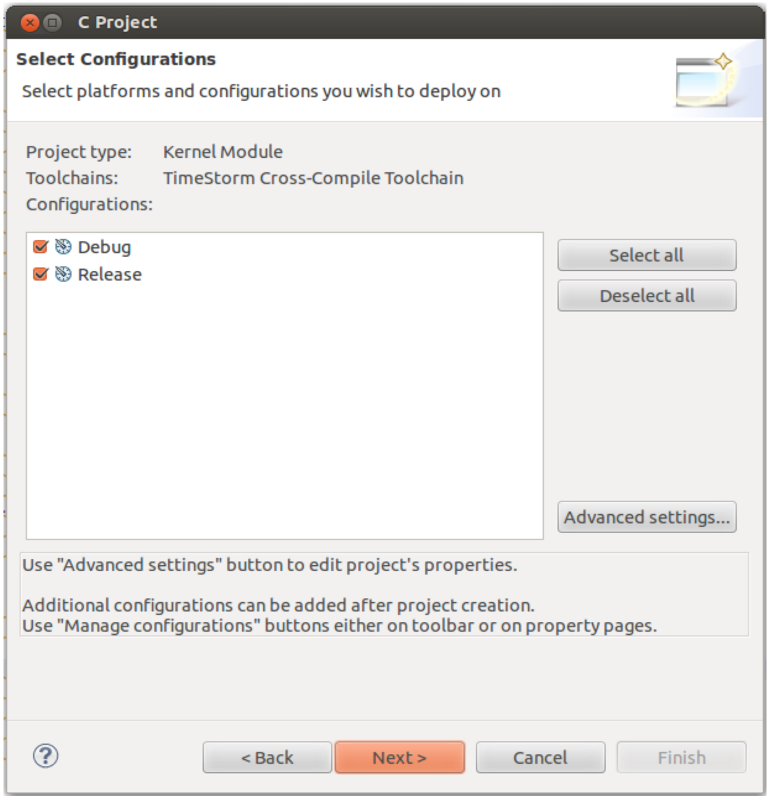
Toolchains: TimeStorm Cross-Compile Toolchain

☑ Show project types and toolchains only if they are supported on the platform

Figure 50: Loadable kernel module project

You can start developing a kernel module project from two of the templates included with TimeStorm, or you can start from an empty project. Enter a name for the project and click next.

Select the build configurations to be created for the project as shown in figure 51 and click Next.



C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Kernel Module
Toolchains: TimeStorm Cross-Compile Toolchain
Configurations:

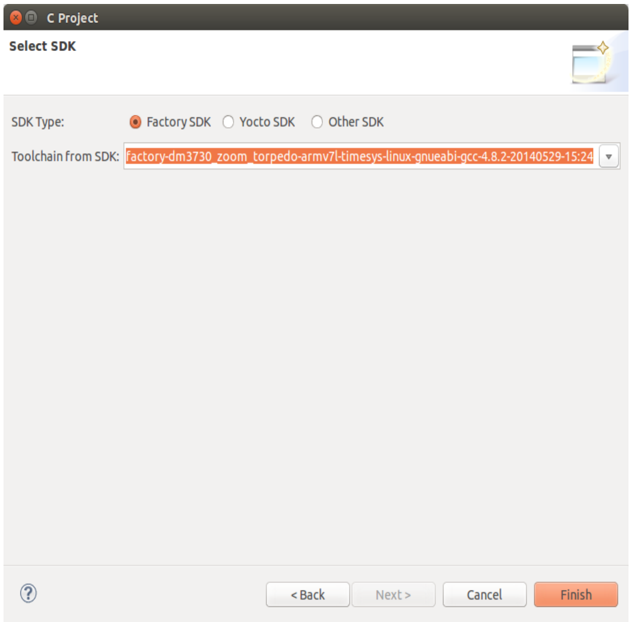
☑ Debug
☑ Release

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

Figure 51: Loadable kernel module project – Build configurations

Select the SDK Type and choose SDK you want to use with the kernel module as shown in *figure 52* and click next. Be sure to select the toolchain that is used with the kernel project / kernel sources you will be choosing on the subsequent page.



C Project

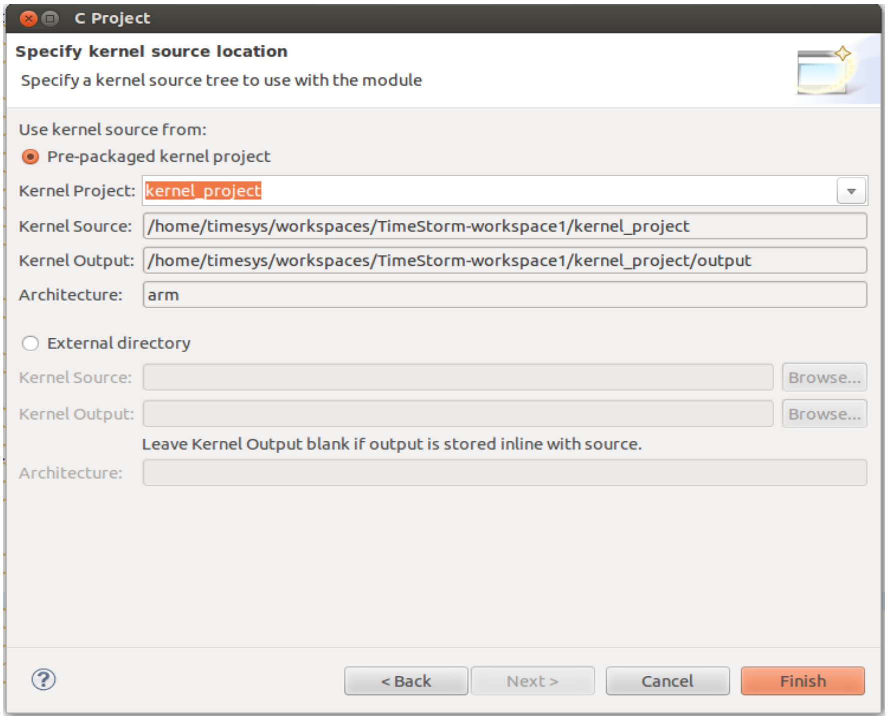
Select SDK

SDK Type: ● Factory SDK ○ Yocto SDK ○ Other SDK

Toolchain from SDK: factory-dm3730_zoom_torpedo-armv7l-timesys-linux-gnueabi-gcc-4.8.2-20140529-15:24

Figure 52: Loadable kernel module project – SDK selection

Kernel module development requires the location of the sources and a kernel image. You can choose a kernel project that is already created in your workspace (recommended) or you can choose an external directory that contains the kernel sources as shown in *figure 53.*



C Project

Specify kernel source location

Specify a kernel source tree to use with the module

Use kernel source from:

● Pre-packaged kernel project

Kernel Project: kernel_project

Kernel Source: /home/timesys/workspaces/TimeStorm-workspace1/kernel_project

Kernel Output: /home/timesys/workspaces/TimeStorm-workspace1/kernel_project/output

Architecture: arm

○ External directory

Kernel Source: Browse...

Kernel Output: Browse...

Leave Kernel Output blank if output is stored inline with source.

Architecture:

Figure 53: Loadable kernel module project – Kernel source location

Click finish to create the kernel module project.

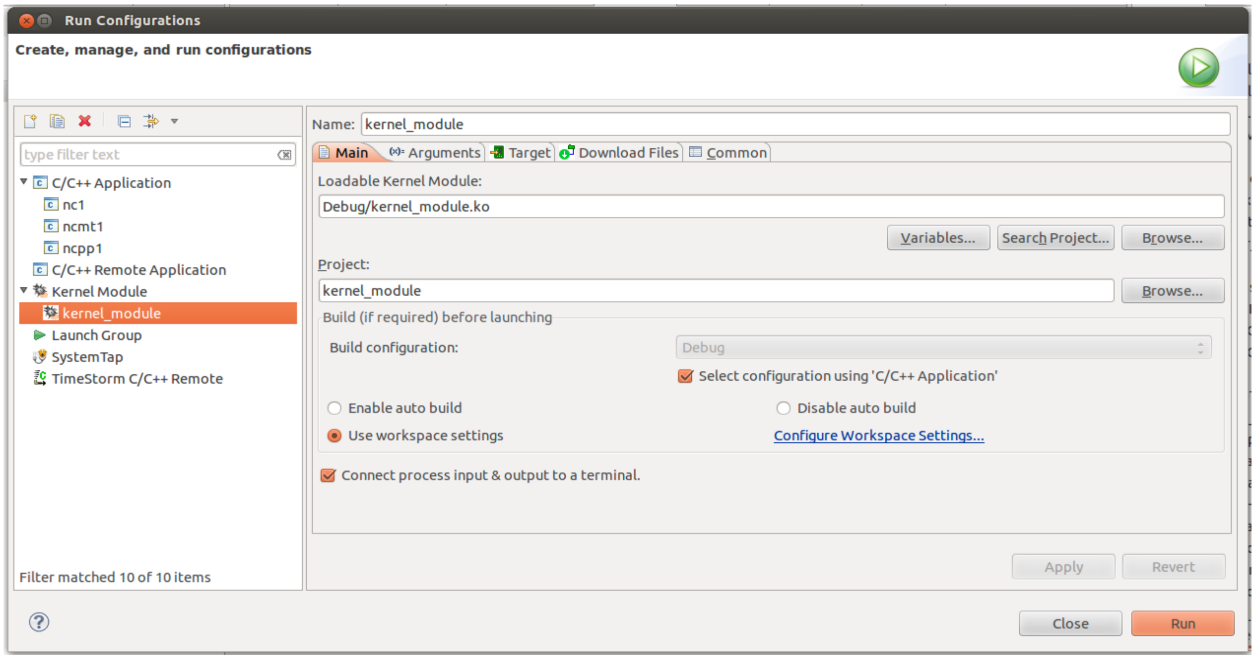
Building Kernel Module Project

A loadable kernel module project is built the same way as other C projects. To build the kernel module, choose a build option from Project menu or from the context menu that appears when you right click on the project name.

Deploying the Kernel Module

To deploy a kernel module on a running target, you will create and run a kernel module launch configuration. To create a kernel module launch configuration, select the kernel module project and click Run > Run configurations or right click on the kernel module project and click Run As > Run Configurations.

Click Kernel Module in the left panel and click New icon in the top left corner.



Name: kernel_module

Loadable Kernel Module: Debug/kernel_module.ko

Project: kernel_module

Build configuration: Debug

☑ Select configuration using 'C/C++ Application'

○ Enable auto build

○ Disable auto build

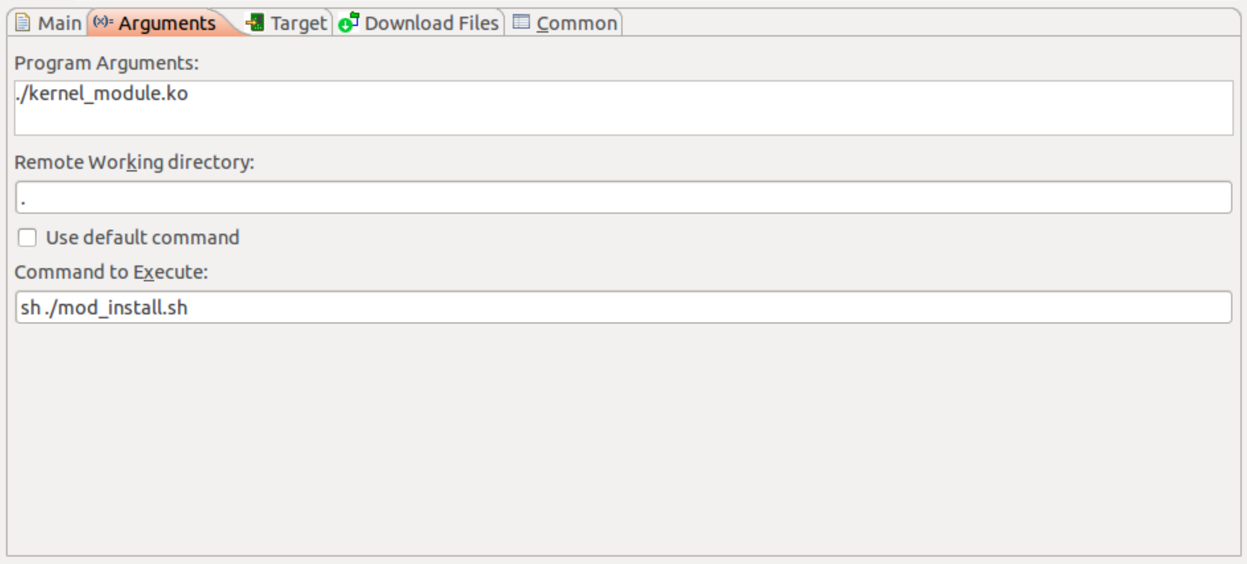
● Use workspace settings

☑ Connect process input & output to a terminal.

Figure 54: Kernel Module launch configuration

Enter a name for the launch configuration. The module to be deployed to the target and the project name are filled in. You can change them if you want to make any modifications.

The kernel module is deployed to the target by using a mod_install.sh script. This script is filled in as the command to execute on the arguments tab. The kernel module is the argument to this script, which is also filled in the program arguments text field.



Program Arguments: ./kernel_module.ko

Remote Working directory: .

☐ Use default command

Command to Execute: sh ./mod_install.sh

Figure 55: Kernel module launch configuration – Arguments tab

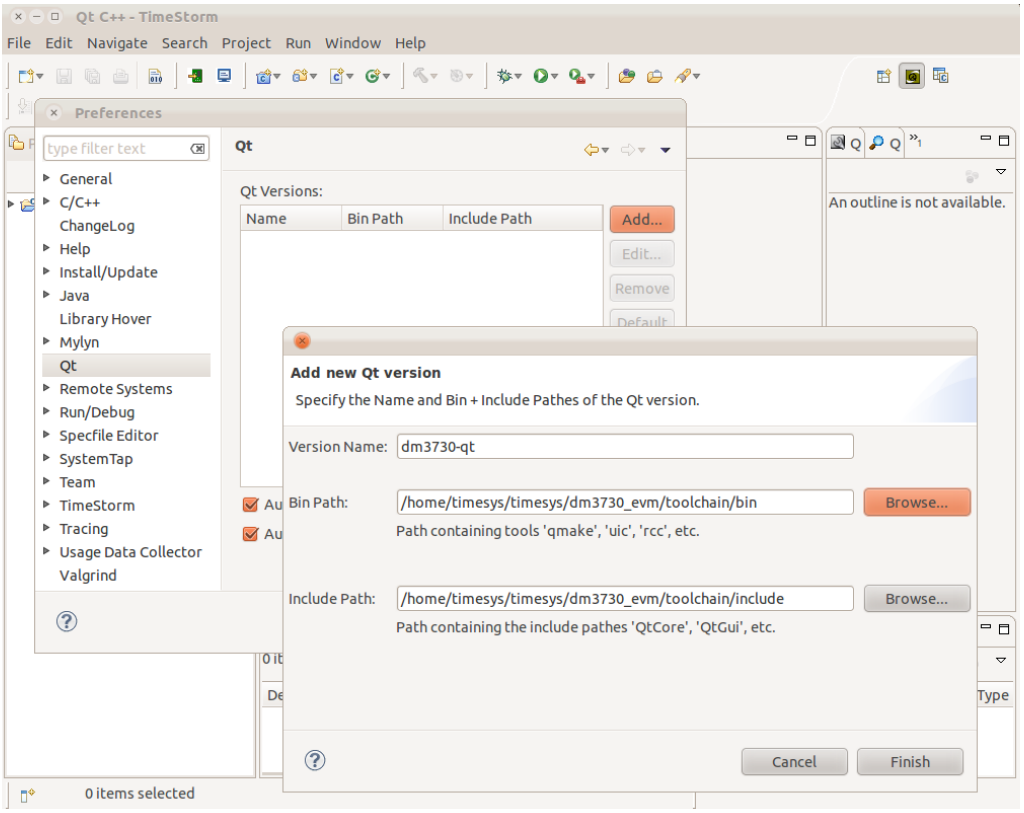
Other tabs in the launch configuration are similar to those explained in the C/C++ application debugging.

Qt Development with TimeStorm

Qt is a cross-platform application and UI framework with APIs for C++ programming. TimeStorm is bundled with the Qt Eclipse integration plugins. These plugins will help in developing Qt applications using a Timesys SDK, and remotely running and debugging the Qt applications on target. Timesys SDKs that include Qt are configured to work easily with TimeStorm. These are a few main steps that get you started developing Qt applications using TimeStorm:

1. **Switch to Qt perspective:** Click *Window > Open Perspective > Other > Qt C++*

2. **Configure TimeStorm for Qt:** To create Qt projects and develop Qt applications, you have to configure TimeStorm for Qt by adding the Qt version you want to use. You can add multiple Qt versions to TimeStorm – native Qt, Qt specific to your board. To add a Qt version, click *Window > Preferences > Qt > Add*. A dialog *(as shown below in Figure 56)* is opened.



Add new Qt version

Specify the Name and Bin + Include Pathes of the Qt version.

Version Name: dm3730-qt

Bin Path: /home/timesys/timesys/dm3730_evm/toolchain/bin

Path containing tools 'qmake', 'uic', 'rcc', etc.

Include Path: /home/timesys/timesys/dm3730_evm/toolchain/include

Path containing the include pathes 'QtCore', 'QtGui', etc.

Figure 56: Configuring Qt

3. **Enter a name for the Qt version**

4. **Select the Bin Path** — Click ‘Browse’ to select the bin folder of the toolchain that includes Qt.

5. **Include Path** — The include path will be automatically set: you need not change it.

6. **Click the ‘Finish’ Button.** Now TimeStorm is set to use the Qt version you have added.

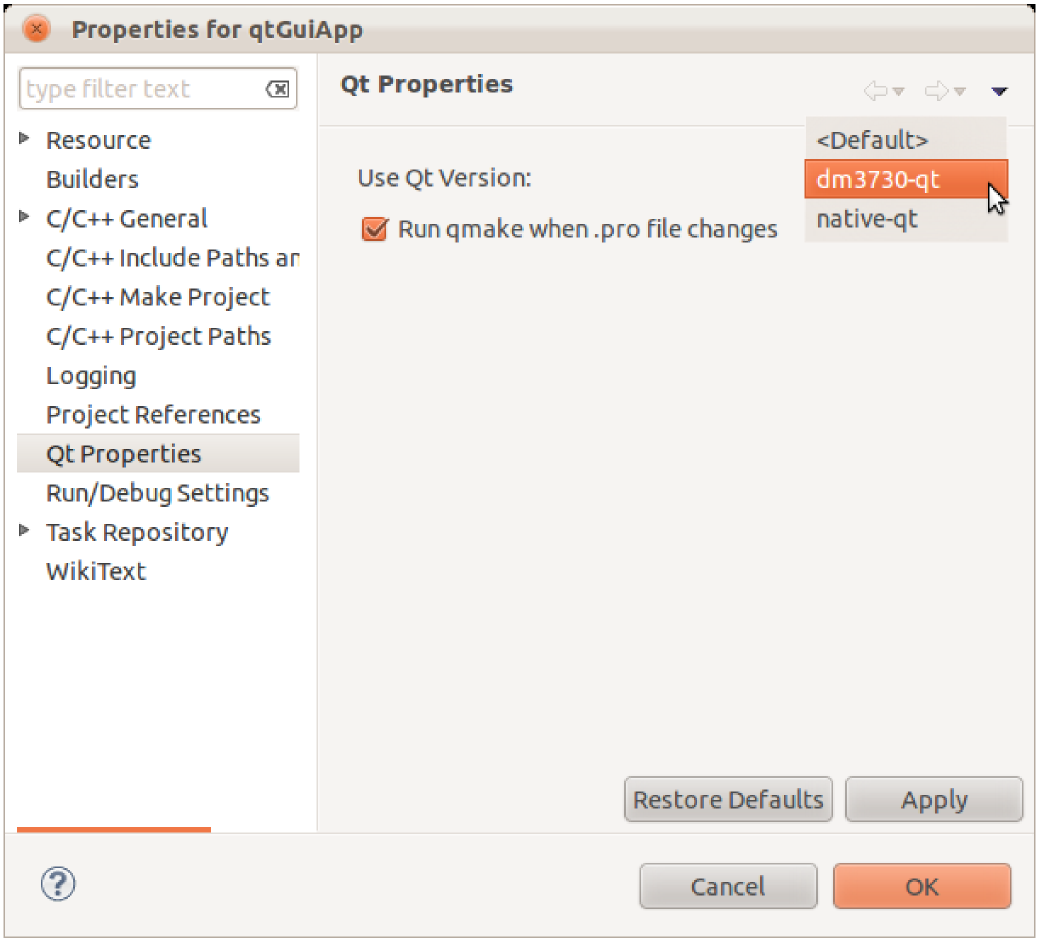
To Create a Qt project and develop a Qt application:

1. Click *File > New > Project > Qt > Qt Console Project / Qt Gui Project* to launch the project wizard
2. Fill in the project name and other details
3. Click the ‘Finish’ button.

During development, you may want to first develop and debug your Qt apps on the host and then rebuild them for the target board.

To switch the Qt version:

1. Right-click on the project, and click *Properties > Qt Properties.*
2. Select the Qt version you want to use *(as shown in Figure 57).*



Use Qt Version: dm3730-qt

☑ Run qmake when .pro file changes

Figure 57: Changing Qt version for a project

After booting the board with the SDK that includes Qt, you will be able to remotely run / debug the Qt apps on the Hardware target.

To Run / Debug Qt an application on the Target:

1. Follow the procedure explained in C/C++ projects.

Working with Yocto SDK

TimeStorm supports and simplifies application development using Yocto (https://www.yoctoproject.org) SDK. TimeStorm automatically detects Yocto SDKs that are installed and supports using Yocto SDK for:

- creating C/C++ applications
- deploying, running and debugging applications on remote hardware targets
- creating, configuring, building and debugging a Kernel project
- creating and deploying Loadable Kernel Modules
- running and testing applications in an emulated environment using QEMU

Setting up Yocto SDK

Yocto SDK components can be downloaded from http://downloads.yoctoproject.org or can be built on your local host. The local build directory on your host is referred below as `<poky_dir_path>`. You may refer to http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html to learn how to build a Yocto SDK.

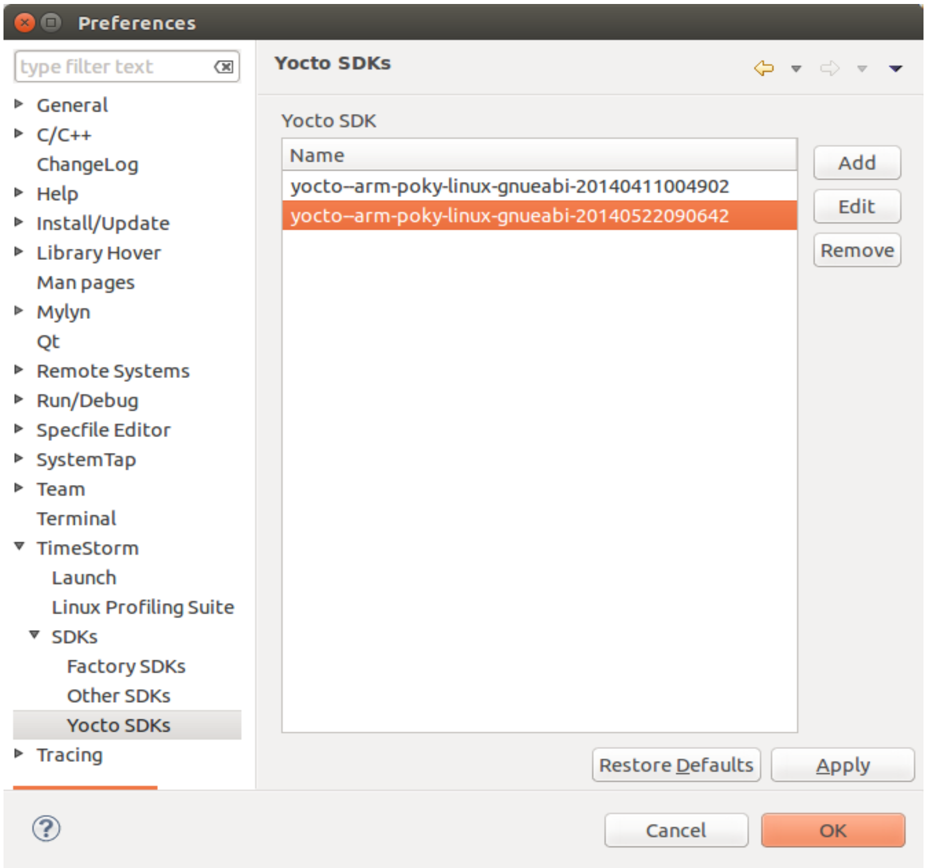
You will need:

- **Yocto toolchain for application development**. Toolchain installer can be downloaded from http://downloads.yoctoproject.org/releases/yocto/yocto-1.6/toolchain/ or can be found in `<poky_dir_path>/build/tmp/deploy/sdk` folder. `chmod +x <toolchain installer>` and `./<toolchain installer>` to install the toolchain. The toolchain is installer in `/opt/poky` directory and automatically detected by TimeStorm. If TimeStorm is running while installing the toolchain, then restart TimeStorm for the toolchain to be detected.
- **Yocto RFS and kernel images to boot a board or run the QEMU emulator.** These can be downloaded from http://downloads.yoctoproject.org/releases/yocto/yocto-1.6/machines/ or can be found in `<poky_dir_path>/build/tmp/deploy/images/` folder.
- **Yocto Kernel sources for create, configure and build a kernel project.** Kernel sources for your specific board cannot be directly downloaded. After building a Yocto BSP, they can be located in `<poky_dir_path>/build/tmp/work/<machine name>-poky-linux-<xxxx>/linux-yocto/<kernel-version>……/linux.`

Yocto SDKs in TimeStorm

Auto detected SDK

TimeStorm automatically detects toolchains that are installed in default location `/opt/poky/` and registers it as a Yocto SDK. You can view the Yocto SDKs by clicking Window > Preferences > TimeStorm > SDKs > Yocto SDK as shown in *figure 58.*



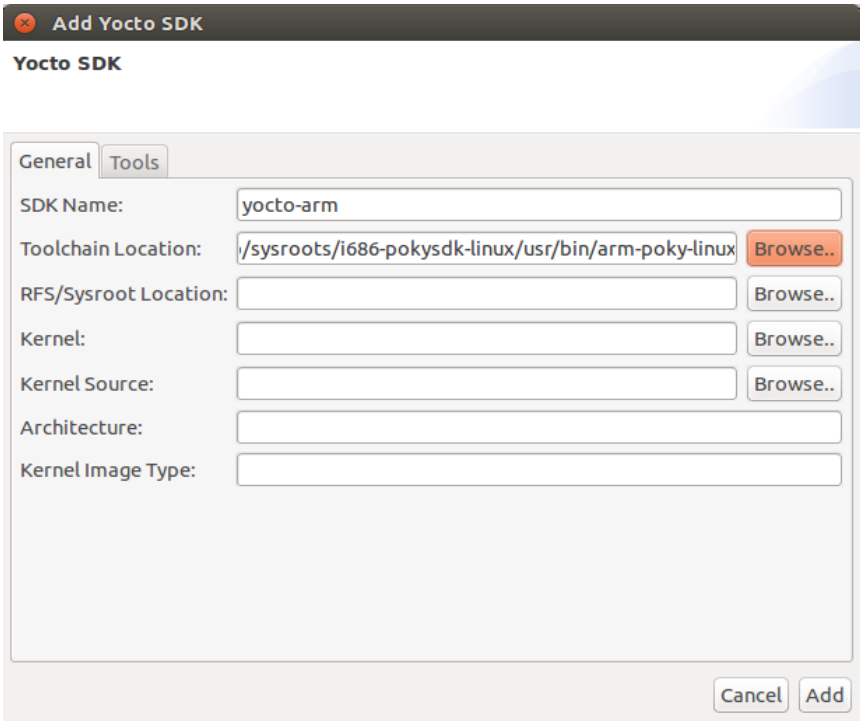
yocto--arm-poky-linux-gnueabi-20140411004902
yocto--arm-poky-linux-gnueabi-20140522090642

Figure 58: Yocto SDKs

Please note that the auto detection of Yocto SDK is limited to the toolchain installed in `/opt/poky` directory. The toolchain is sufficient for C / C++ projects. For using QEMU, and for creating Kernel and Kernel module projects you have to edit the Yocto SDK and point to the Kernel Image, Kernel Sources and the RFS. To Edit a Yocto SDK, click Window > Preferences > TimeStorm > SDKs > Yocto SDK, select the Yocto SDK you want to edit and click the Edit button to open the 'Edit Yocto SDK' dialog. In this dialog you can use the browse button to locate the RFS/Sysroot location (.ext3), Kernel Image (.bin) and Kernel Sources (folder that has folders named arch, drivers, kernel etc.).

Manually adding an SDK

If you have installed the Yocto toolchain in a non-default location, you can manually add it to TimeStorm by clicking Window > Preferences > TimeStorm > SDKs > Yocto SDK and clicking the ‘Add’ button. This will open a ‘Add Yocto SDK’ dialog as shown in *figure 59*. You have to just enter the SDK name and toolchain location for C/C++ application development. All other fields are optional. You will need the RFS/Sysroot location, kernel and kernel source entries for working with QEMU, Kernel and Kernel Module projects.



Add Yocto SDK

Yocto SDK

SDK Name: yocto-arm

Toolchain Location: /sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux

RFS/Sysroot Location:

Kernel:

Kernel Source:

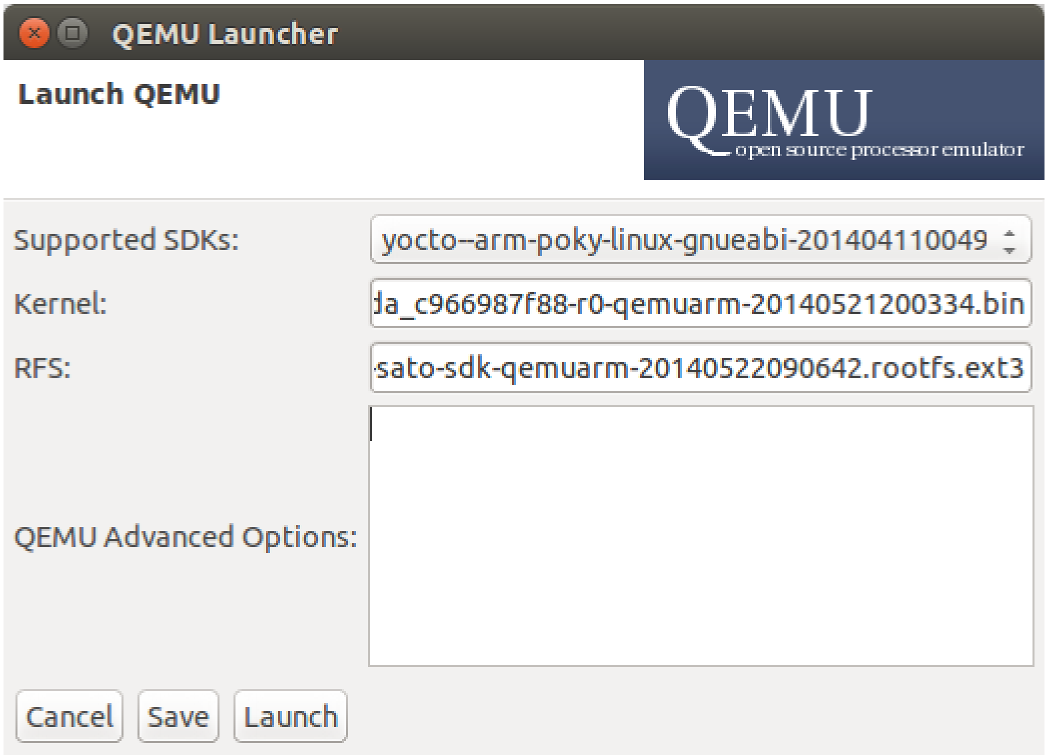
Architecture:

Kernel Image Type:

Figure 59: Add Yocto SDK

QEMU Launcher

QEMU Launcher will allow you to select a QEMU enabled SDK and run QEMU. QEMU launcher dialog can be opened by clicking the ⟳ icon in the toolbar or by clicking Run > Launch QEMU. The launcher dialog will be displayed as show in *figure 60*.



Supported SDKs: yocto--arm-poky-linux-gnueabi-201404110049

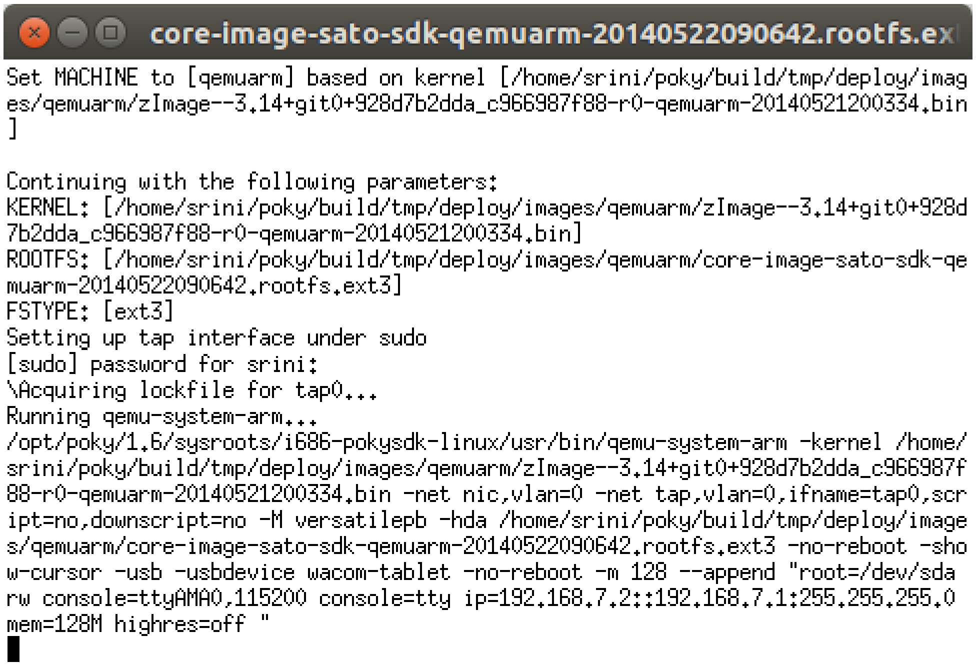
Kernel: da_c966987f88-r0-qemuarm-20140521200334.bin

RFS: sato-sdk-qemuarm-20140522090642.rootfs.ext3

QEMU Advanced Options:

Figure 60: QEMU Launcher Dialog

Launch QEMU: QEMU enabled SDKs are Yocto SDKs that are set with the Kernel Image and RFS/Sysroot location. QEMU launcher will detect the QEMU enabled SDKs and populate them in the Supported SDKs drop down box. To launch QEMU, select an SDK. The kernel and RFS for QEMU are pre filled from the selected SDK. If you want to pass additional options to QEMU, you may enter it in the QEMU Advanced Options text area. Click the launch button to launch QEMU. QEMU will be launched in a terminal window and you will be required to enter your sudo password as shown in *figure 61*.



```
Set MACHINE to [qemuarm] based on kernel [/home/srini/poky/build/tmp/deploy/imag
es/qemuarm/zImage--3.14+git0+928d7b2dda_c966987f88-r0-qemuarm-20140521200334.bin
]

Continuing with the following parameters:
KERNEL: [/home/srini/poky/build/tmp/deploy/images/qemuarm/zImage--3.14+git0+928d
7b2dda_c966987f88-r0-qemuarm-20140521200334.bin]
ROOTFS: [/home/srini/poky/build/tmp/deploy/images/qemuarm/core-image-sato-sdk-qe
muarm-20140522090642.rootfs.ext3]
FSTYPE: [ext3]
Setting up tap interface under sudo
[sudo] password for srini:
\Acquiring lockfile for tap0...
Running qemu-system-arm...
/opt/poky/1.6/sysroots/i686-pokysdk-linux/usr/bin/qemu-system-arm -kernel /home/
srini/poky/build/tmp/deploy/images/qemuarm/zImage--3.14+git0+928d7b2dda_c966987f
88-r0-qemuarm-20140521200334.bin -net nic,vlan=0 -net tap,vlan=0,ifname=tap0,scr
ipt=no,downscript=no -M versatilepb -hda /home/srini/poky/build/tmp/deploy/image
s/qemuarm/core-image-sato-sdk-qemuarm-20140522090642.rootfs.ext3 -no-reboot -sho
w-cursor -usb -usbdevice wacom-tablet -no-reboot -m 128 --append "root=/dev/sda
rw console=ttyAMA0,115200 console=tty ip=192.168.7.2::192.168.7.1:255.255.255.0
mem=128M highres=off "
█
```

Figure 61: QEMU terminal window

After entering the sudo password, QEMU will run in a separate graphical window as shown in *figure 62.* If you wish not to run QEMU in a separate graphical window, but continue to run in the terminal window as in *figure 61*, enter `-nographic` in the advance parameters text field in the launcher dialog.

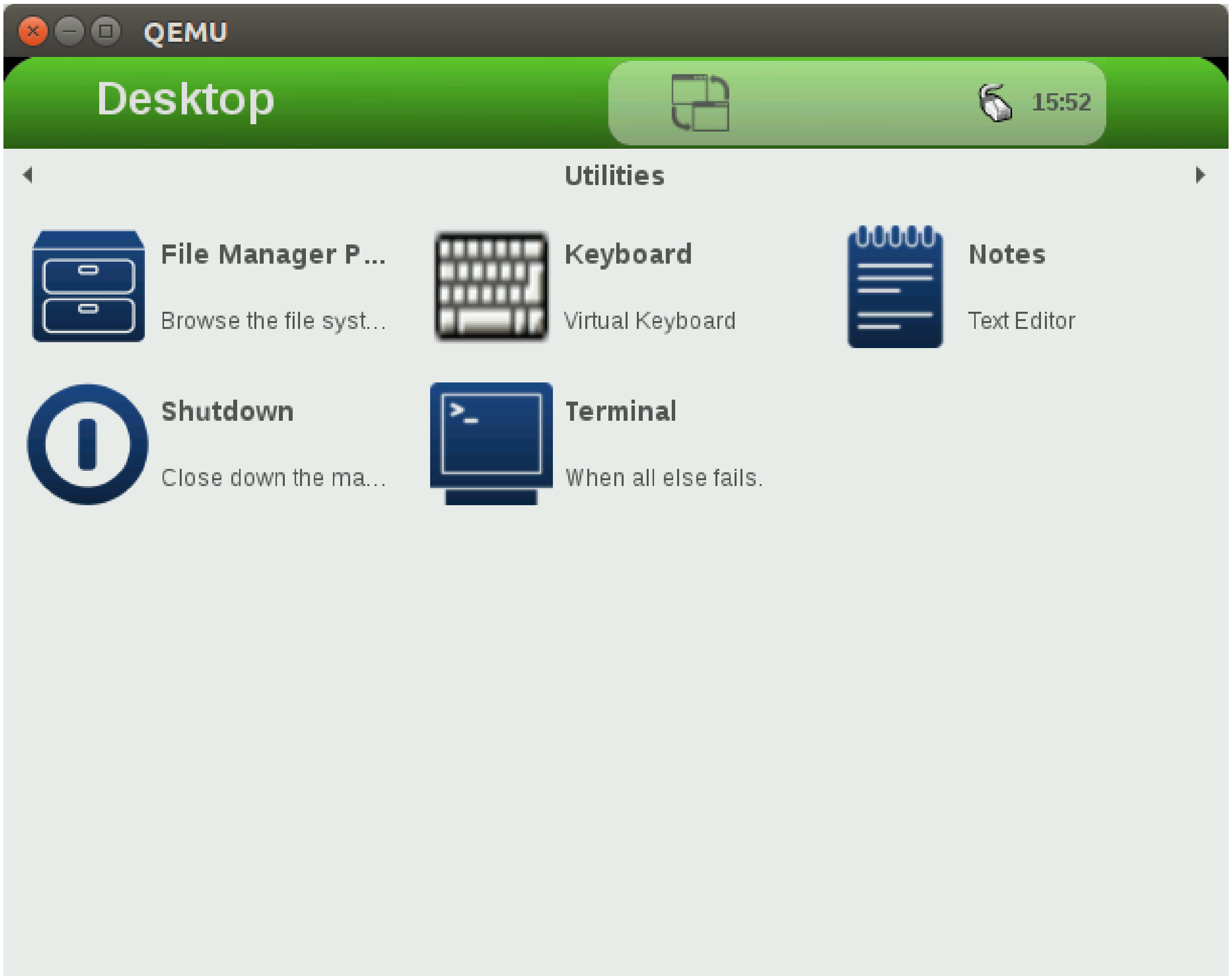


Figure 62: QEMU graphical window

Run / Debug application in QEMU: To run / debug an application in QEMU you have to create a hardware target definition for QEMU. By default, QEMU is assigned an IP address of `192.168.7.2`. Set the root password for QEMU, and use them to define a hardware target as explained in Working with Hardware Targets. Then you can use this QEMU to run / debug your application.

Stopping QEMU: To stop QEMU type `reboot` in the QEMU window. You may have to enter the sudo password again in the QEMU console. You can close the QEMU console window after QEMU has terminated.

Remote System Explorer

Remote System Explorer (or RSE in short) is the new addition from Eclipse for remote target interaction. RSE allows you to define targets, explore target file system, and run and debug applications on target. Timesys recommends using the 'Hardware Target' functionality provided by Timesys as explained in the Working with Hardware Targets section, because:

- RSE using SSH does not work with dropbear
- RSE does not support serial connection
- RSE does not support NFS

To use RSE with boards running Timesys SDK:

- Include openssh package in your target RFS. RSE requires a sftp server on the target and the dropbear ssh server does not include an sftp server.
- When defining a connection in RSE, choose 'SSH only' system type.

After you define a target connection in RSE, you will be able to browse the target file system, copy files between local host and target, and open a terminal console to the target.

For additional information on RSE, refer to the Help included in TimeStorm by selecting *Help > Help Contents*.

Linux Profiling Suite (LPS)

TimeStorm supplements the Eclipse OProfile tool with its own Linux Profiling Suite (LPS). LPS is easy to use and well integrated with TimeStorm hardware targets management.

LPS allows embedded developers to find performance bottlenecks in their code. LPS uses OProfile, which is Open Source profiler for Linux systems. OProfile collects information about kernel and all running applications. OProfile profiles your code by recording the output from hardware registers that counts events such as CPU cycles. LPS provides statistical analysis of profile results and annotated source code view, helps in identifying any performance issues easily and quickly.

For more information about OProfile, refer to the OProfile Manual found at http://oprofile.sourceforge.net/doc/index.html

Note: You must have OProfile installed on target for remote profiling or on host machine for local profiling.

LPS Perspective

Select *Window > Open Perspective > Other* option from main menu and choose 'Linux Profiling Suite' from 'Open Perspective' window (*shown in figure 63*).

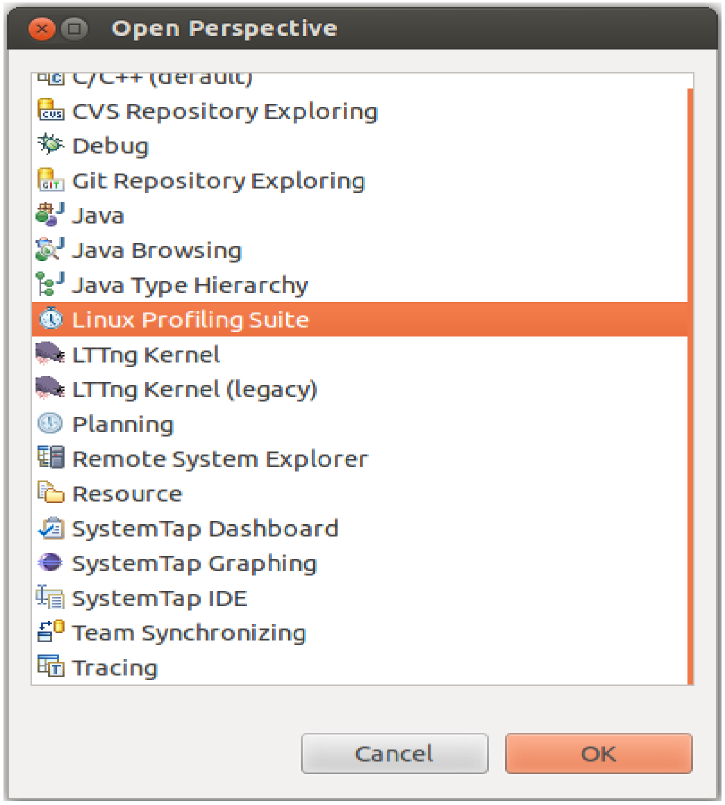
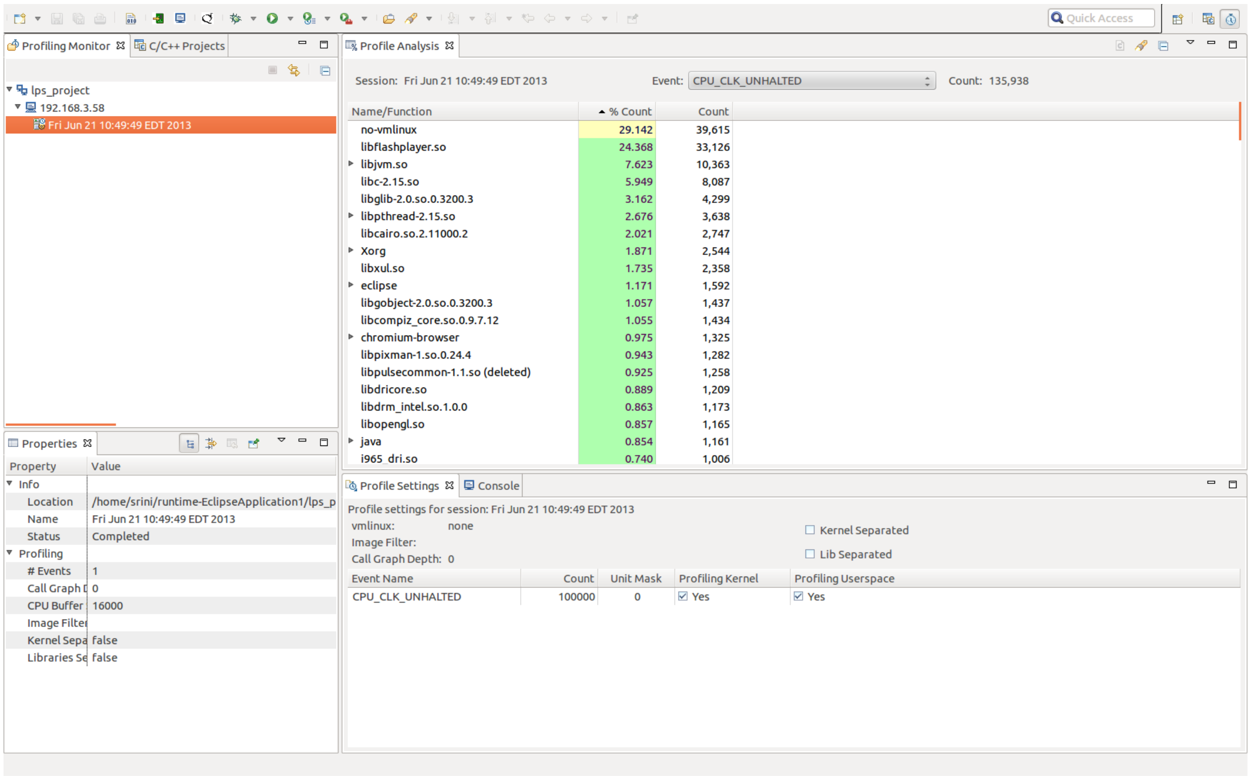


Figure 63: Open Perspective Window

The LPS perspective shown in *figure 64*, includes:

- Profiling Monitor view – to organize and manage profiling sessions
- Profile Analysis view – to view and analysze the OProfile profile data
- Properties view – to view the properties of the actively selected node in the Profiling Monitor View
- Profile Settings view – to view the various OProfile settings used for the profiling session



lps_project
192.168.3.58
Fri Jun 21 10:49:49 EDT 2013

Property	Value
Info	
Location	/home/srini/runtime-EclipseApplication1/lps_p
Name	Fri Jun 21 10:49:49 EDT 2013
Status	Completed
Profiling	
# Events	1
Call Graph D	0
CPU Buffer	16000
Image Filte	
Kernel Sepa	false
Libraries Se	false

Session: Fri Jun 21 10:49:49 EDT 2013 Event: CPU_CLK_UNHALTED Count: 135,938

Name/Function	% Count	Count
no-vmlinux	29.142	39,615
libflashplayer.so	24.368	33,126
libjvm.so	7.623	10,363
libc-2.15.so	5.949	8,087
libglib-2.0.so.0.3200.3	3.162	4,299
libpthread-2.15.so	2.676	3,638
libcairo.so.2.11000.2	2.021	2,747
Xorg	1.871	2,544
libxul.so	1.735	2,358
eclipse	1.171	1,592
libgobject-2.0.so.0.3200.3	1.057	1,437
libcompiz_core.so.0.9.7.12	1.055	1,434
chromium-browser	0.975	1,325
libpixman-1.so.0.24.4	0.943	1,282
libpulsecommon-1.1.so (deleted)	0.925	1,258
libdricore.so	0.889	1,209
libdrm_intel.so.1.0.0	0.863	1,173
libopengl.so	0.857	1,165
java	0.854	1,161
i965_dri.so	0.740	1,006

Profile settings for session: Fri Jun 21 10:49:49 EDT 2013
vmlinux: none
Image Filter:
Call Graph Depth: 0

☐ Kernel Separated
☐ Lib Separated

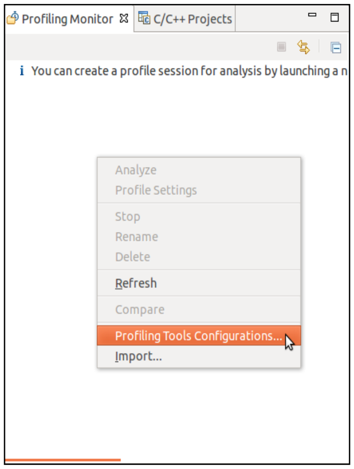
Event Name	Count	Unit Mask	Profiling Kernel	Profiling Userspace
CPU_CLK_UNHALTED	100000	0	☑ Yes	☑ Yes

Figure 64: LPS Perspective

Profiling using LPS

To profile an application or system, create a profile launch configuration.

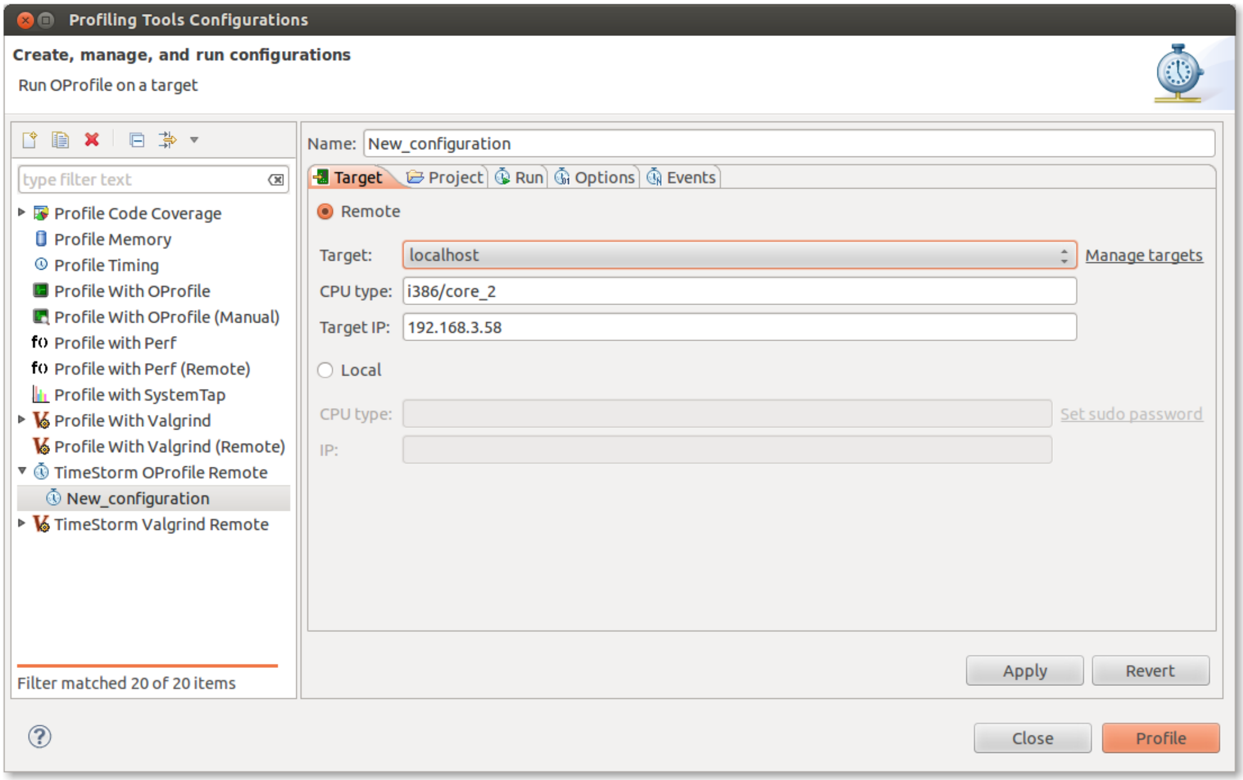
1. In the 'Profiling Monitor' right click and select 'Profiling Tools Configuration' to create 'Profiling Tools Configuration' as shown in *figure 65.*



You can create a profile session for analysis by launching a n

Figure 65: Launching Profiling Tools Configuration

2. Select ‘TimeStorm Oprofile’ in left panel.
3. Click New launch configuration icon in top left corner. This will create a new profile launch configuration and the launch configuration tabs are displayed in the right panel (shown in *figure 66)*



Profiling Tools Configurations

Create, manage, and run configurations

Run OProfile on a target

Name: New_configuration

● Remote

Target: localhost

CPU type: i386/core_2

Target IP: 192.168.3.58

○ Local

CPU type:

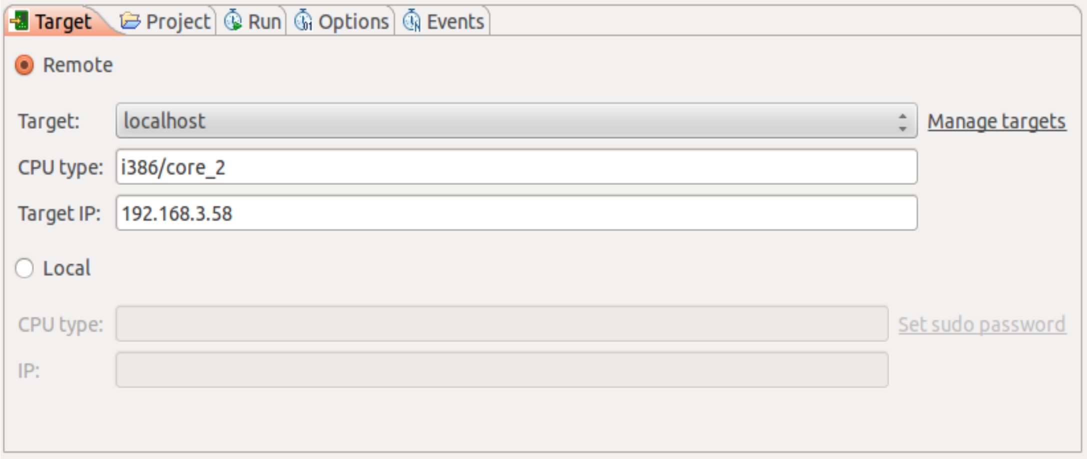
IP:

Filter matched 20 of 20 items

Figure 66: Creating a profile launch configuration

Target Tab

The target tab is show in *figure 67.* Choose “Remote” to run OProfile on remote target or choose “Local” to run OProfile on local host. When the target is selected, TimeStorm checks the OProfile CPU type and fills in the details. An error message is displayed if TimeStorm cannot run OProfile commands.



● Remote

Target: localhost

CPU type: i386/core_2

Target IP: 192.168.3.58

○ Local

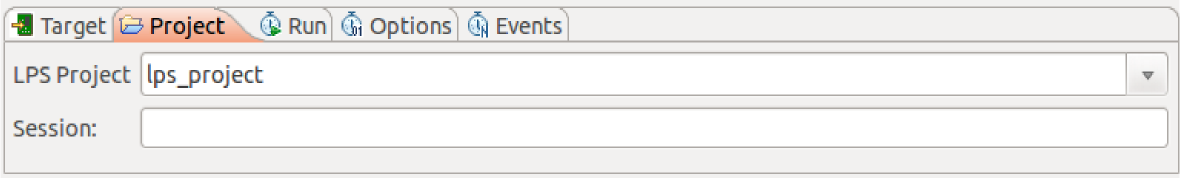
CPU type:

IP:

Figure 67: LPS Configuration – Target Tab

Project Tab

The project tab is shown in *figure 68.* LPS organizes profiling sessions into projects and sessions. You can create a new profiling project or choose an existing one. The session name is optional. If session name is left blank, TimeStorm creates a session using the date and time.



LPS Project lps_project

Session:

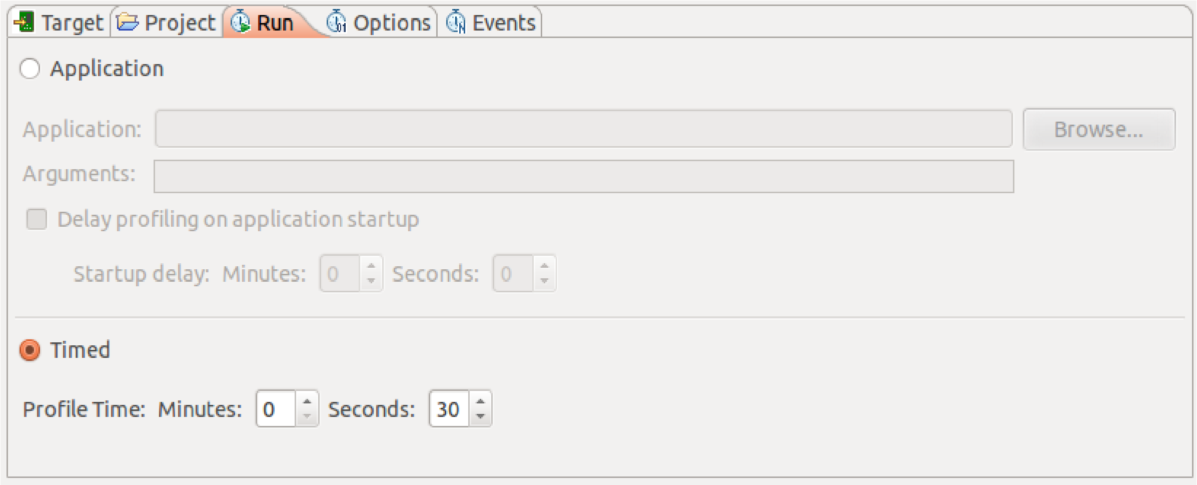
Figure 68: LPS Configuration – Project Tab

Run Tab

The run tab is shown in *figure 69.* LPS allows you to profile an application or the whole system for a specified duration.

Application – If you want to profile an application, select it from either your workspace or file system by clicking the Browse button. If you want to skip profiling during application startup, you can delay the OProfile startup by setting the delay time.

Timed – Select this option to profile the whole system for specific duration.



○ Application

Application:

Arguments:

□ Delay profiling on application startup

Startup delay: Minutes: 0 Seconds: 0

● Timed

Profile Time: Minutes: 0 Seconds: 30

Figure 69: LPS Configuration – Run Tab

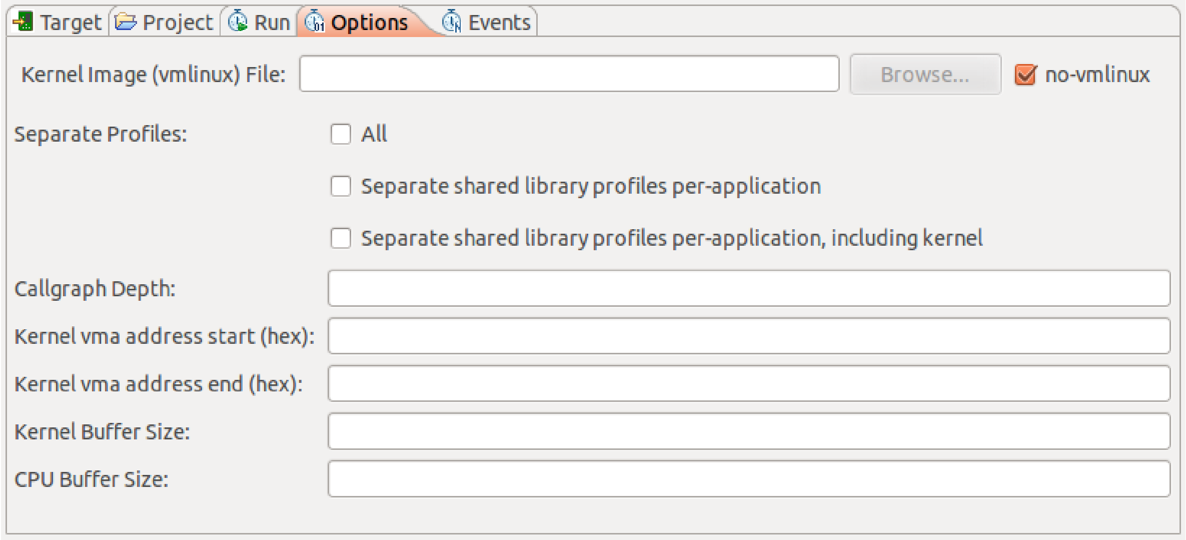
Options Tab

This tab allows you to configure OProfile.

Kernel image (vmlinux) – If you want to profile the kernel, select the vmlinux kernel image file by clicking browse. If you do not have a vmlinux file or you do not want to profile the kernel, select 'no-vmlinux' checkbox.

For a detailed explanation of other options on this tab, please refer to:

http://oprofile.sourceforge.net/doc/controlling-daemon.html



Kernel Image (vmlinux) File: ☑ no-vmlinux

Separate Profiles: □ All

□ Separate shared library profiles per-application

□ Separate shared library profiles per-application, including kernel

Callgraph Depth:

Kernel vma address start (hex):

Kernel vma address end (hex):

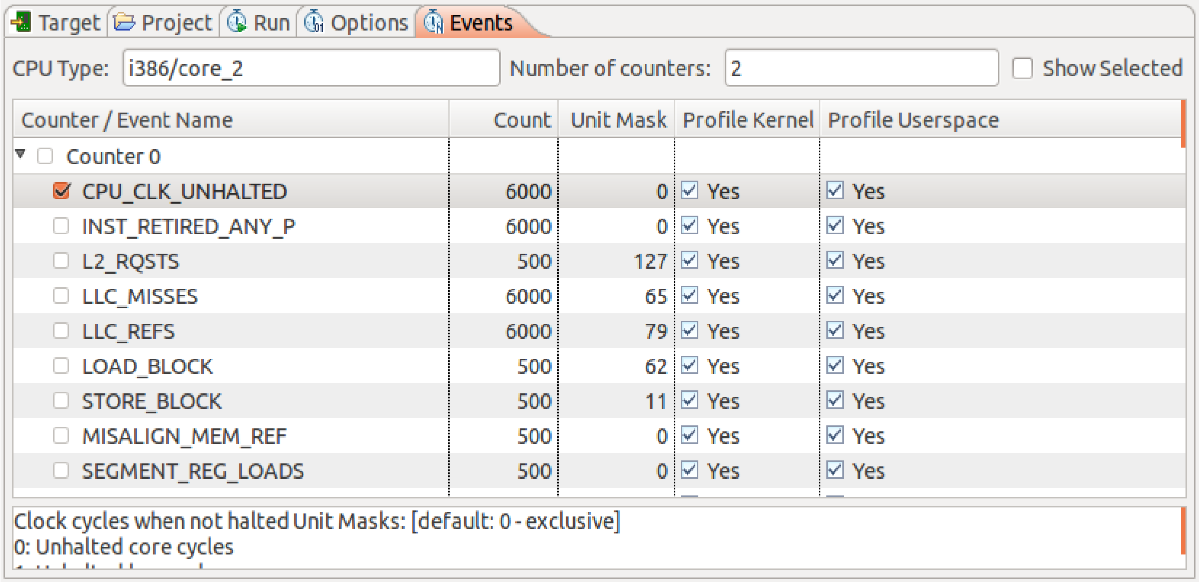
Kernel Buffer Size:

CPU Buffer Size:

Figure 70: LPS Configuration – Options Tab

Events Tab

This tab allows you to specify the events for each of the hardware performance counters. The OProfile CPU type and the number of performance counters on your target are displayed at the top.



CPU Type: i386/core_2 Number of counters: 2 □ Show Selected

Counter / Event Name	Count	Unit Mask	Profile Kernel	Profile Userspace
▾ □ Counter 0				
☑ CPU_CLK_UNHALTED	6000	0	☑ Yes	☑ Yes
□ INST_RETIRED_ANY_P	6000	0	☑ Yes	☑ Yes
□ L2_RQSTS	500	127	☑ Yes	☑ Yes
□ LLC_MISSES	6000	65	☑ Yes	☑ Yes
□ LLC_REFS	6000	79	☑ Yes	☑ Yes
□ LOAD_BLOCK	500	62	☑ Yes	☑ Yes
□ STORE_BLOCK	500	11	☑ Yes	☑ Yes
□ MISALIGN_MEM_REF	500	0	☑ Yes	☑ Yes
□ SEGMENT_REG_LOADS	500	0	☑ Yes	☑ Yes

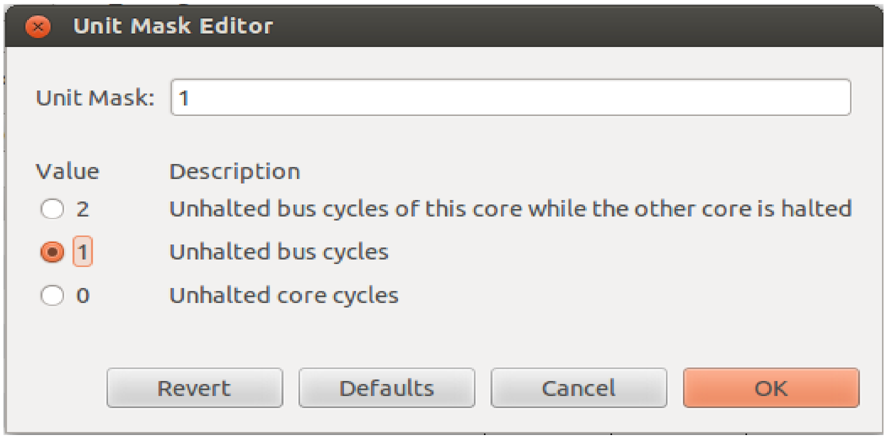
Clock cycles when not halted Unit Masks: [default: 0 - exclusive]
0: Unhalted core cycles

Figure 71: LPS Configuration – Events Tab

Counter / Event Name – Expand the counter to view events in the counter. Select the event for profiling. You may only choose one event per counter. Also, if you choose a particular event for one counter, you may not choose the same event for another counter.

Count – The field shows the counter reset value for the given event. To edit this value, click on counter value, type in new value and hit Enter key.

Unit Mask – This field shows the unit mask. To edit this value, select the event and enter new mask value in 'Unit Mask Editor' (*shown in figure 72)*



Unit Mask: 1

Value	Description
○ 2	Unhalted bus cycles of this core while the other core is halted
● 1	Unhalted bus cycles
○ 0	Unhalted core cycles

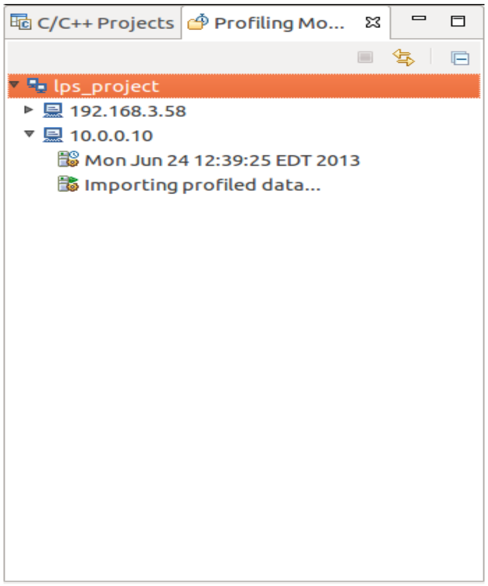
Figure 72: Unit Mask Editor

Profile Kernel – Select this to profile kernel code.

Profile Userspace – Select this to profile userspace code.

For more information on the counters and events, please refer to OProfile documentation.

After configuring OProfile, 'Click' the Profile button to start profiling. The profiling status and progress is displayed in the Profiling monitor view.



```
▾ lps_project
  ▸ 192.168.3.58
  ▾ 10.0.0.10
      Mon Jun 24 12:39:25 EDT 2013
      Importing profiled data...
```

Figure 73: Status messages in Profiling Monitor view

Analyzing the Profiled Data

To view and analyze a successfully completed profiling session, right click on the session and click on "Analyze" as shown in *figure 74.* Alternatively, you can also double click on the session. This will open the profile analysis view and display the profiled data.

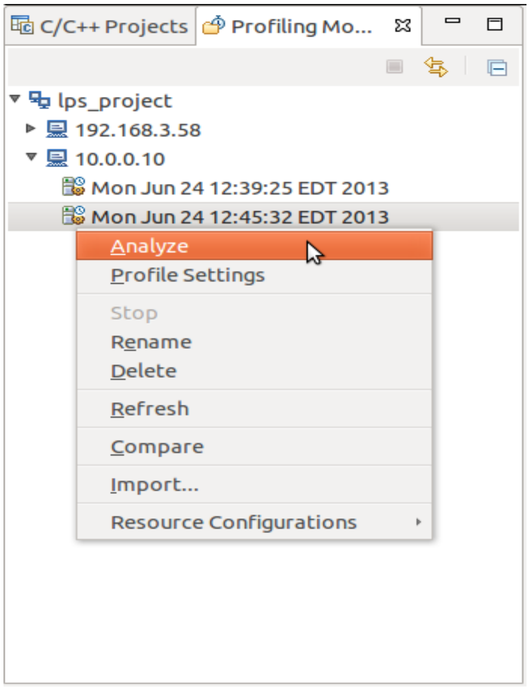
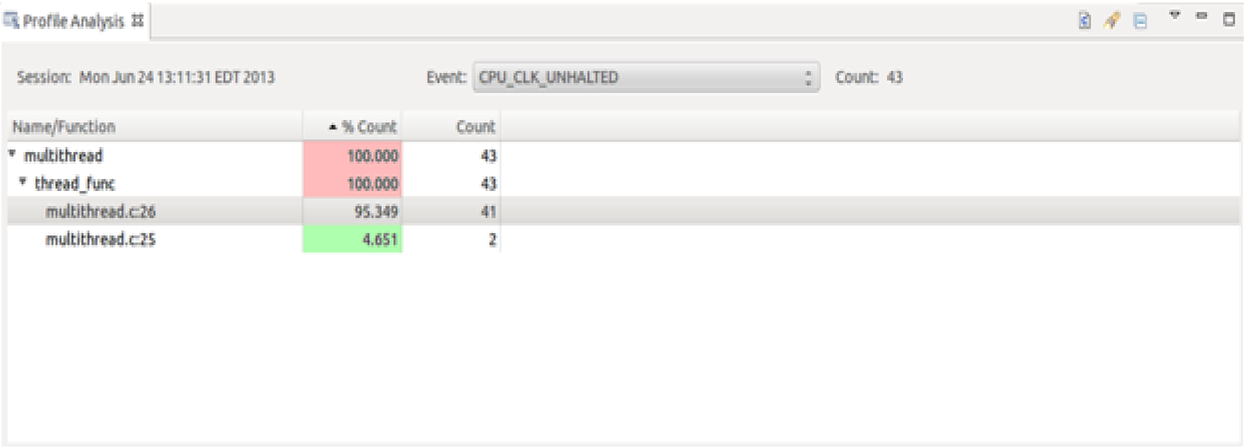


Figure 74: Selecting Analyze option

The profile analysis view displays the session name and the profiled events in a drop down. Selecting a profile event will display the total profile count for that event and the applications, libraries and functions that are profiled for that event. When an executable includes debug information, the filename and the line number are also displayed. For example in the figure, CPU_CLK_HALTED event is selected, multithread application is profiled, the thread_func has the highest profile count at line number 26.



Session: Mon Jun 24 13:11:31 EDT 2013

Event: CPU_CLK_UNHALTED

Count: 43

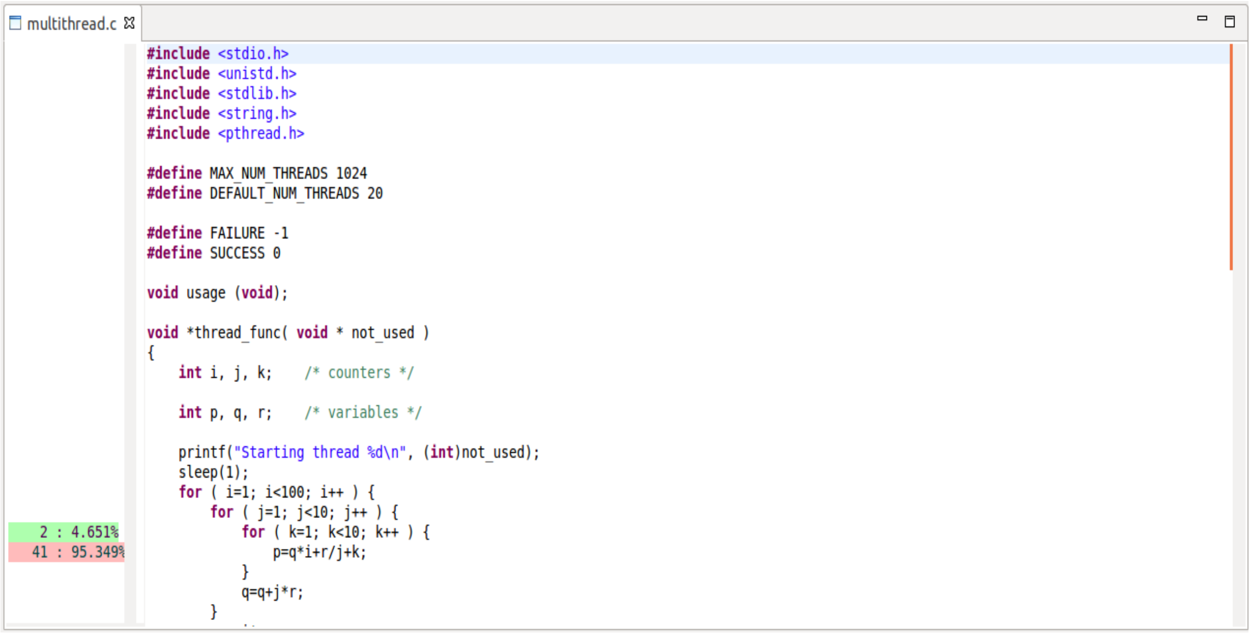
Name/Function	▲ % Count	Count
▾ multithread	100.000	43
▾ thread_func	100.000	43
multithread.c:26	95.349	41
multithread.c:25	4.651	2

Figure 75: LPS Profile Analysis view

To analyze and drill down through large profile data, you can

- sort by column by clicking on the column header
- search in the filename / function by using the search facility by clicking flashlight icon in the analysis view toolbar.
- ▽ Focus on the functions of interest or critical bottle necks by using the name filter, count filter and percentage count filter. The filters can be configured, set and unset using the down arrow in the analysis view toolbar.

For files that have debug information, you can view the profile data and the source code side by side in the annotated source editor as shown in *figure 76.* You can open the annotated source editor by clicking the ‘View Source’ button in analysis view’s toolbar.

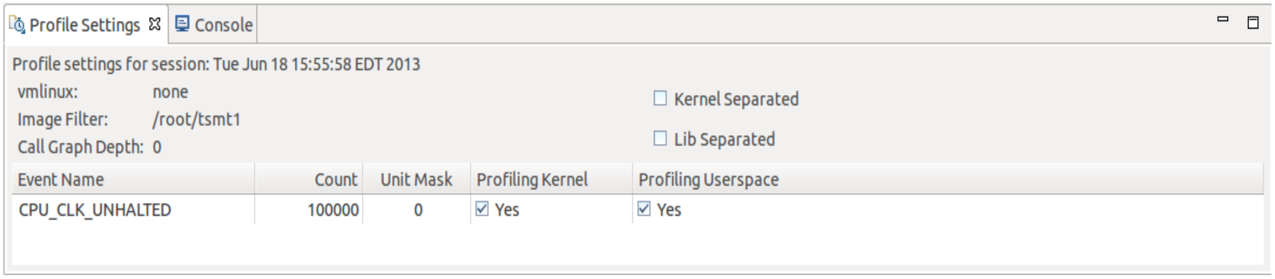


```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

#define MAX_NUM_THREADS 1024
#define DEFAULT_NUM_THREADS 20

#define FAILURE -1
#define SUCCESS 0

void usage (void);

void *thread_func( void * not_used )
{
    int i, j, k;    /* counters */

    int p, q, r;    /* variables */

    printf("Starting thread %d\n", (int)not_used);
    sleep(1);
    for ( i=1; i<100; i++ ) {
        for ( j=1; j<10; j++ ) {
            for ( k=1; k<10; k++ ) {
                p=q*i+r/j+k;
            }
            q=q+j*r;
        }
```

Figure 76: LPS – Annotated C Editor

Profile Settings View

To view the OProfile settings for a profile session, right-click on the session in profiling monitor view and select 'Profile Settings'. The settings are displayed in Profile Settings view as shown in *figure 77.*



Profile settings for session: Tue Jun 18 15:55:58 EDT 2013

vmlinux: none

Image Filter: /root/tsmt1

Call Graph Depth: 0

☐ Kernel Separated

☐ Lib Separated

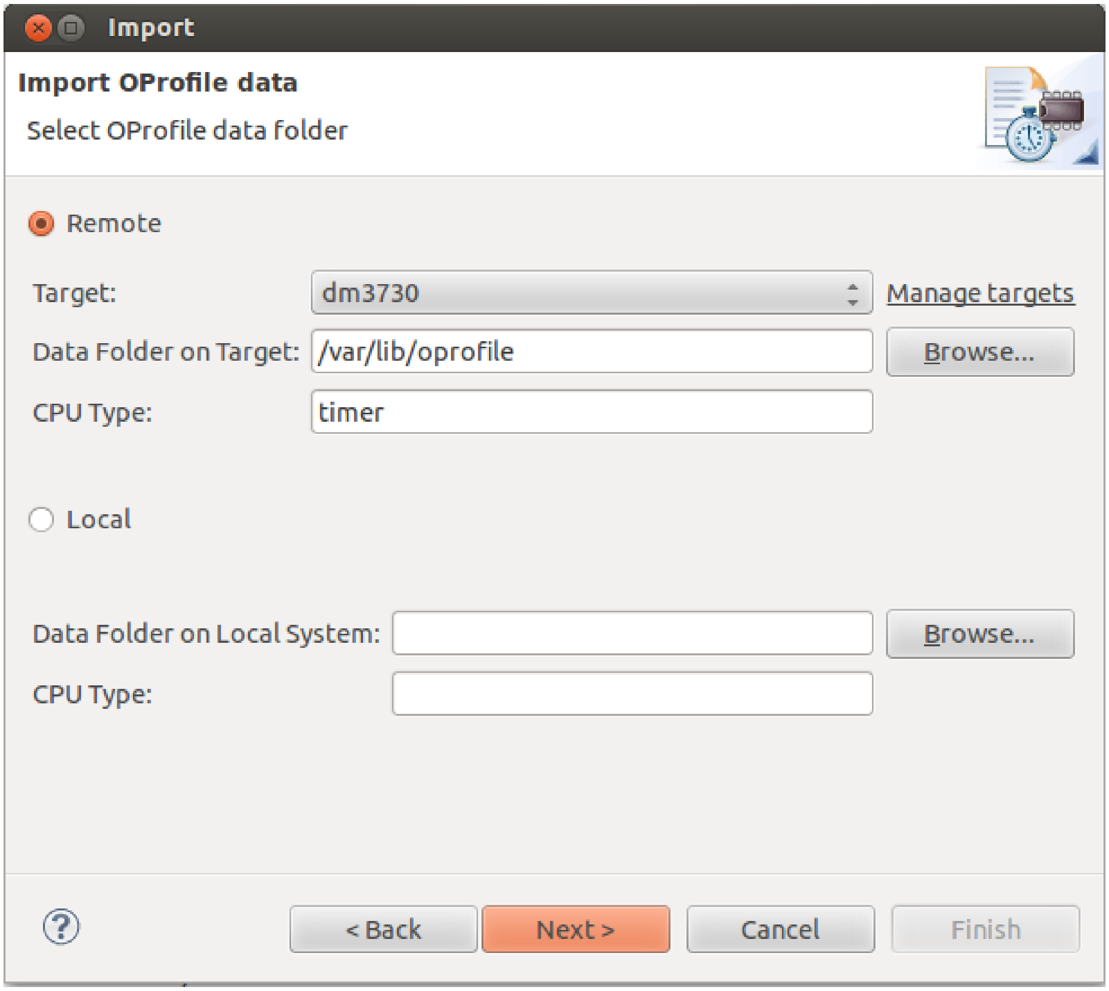
Event Name	Count	Unit Mask	Profiling Kernel	Profiling Userspace
CPU_CLK_UNHALTED	100000	0	☑ Yes	☑ Yes

Figure 77: LPS – Profile Settings view

Importing OProfile Data

If you have already profiled your system / application using OProfile and you want to use LPS to analyze the results, you can import the data to create a session for analysis. To import OProfile data,

1. Right click in the 'Profiling Monitor' view and select the 'Import' option from context menu to open Import wizard.
2. Choose *Other > OProfile Data into LPS Project* and then click 'Next' to open the Import dialog (*shown in figure 78*)



Import

Import OProfile data

Select OProfile data folder

● Remote

Target: dm3730

Data Folder on Target: /var/lib/oprofile

CPU Type: timer

○ Local

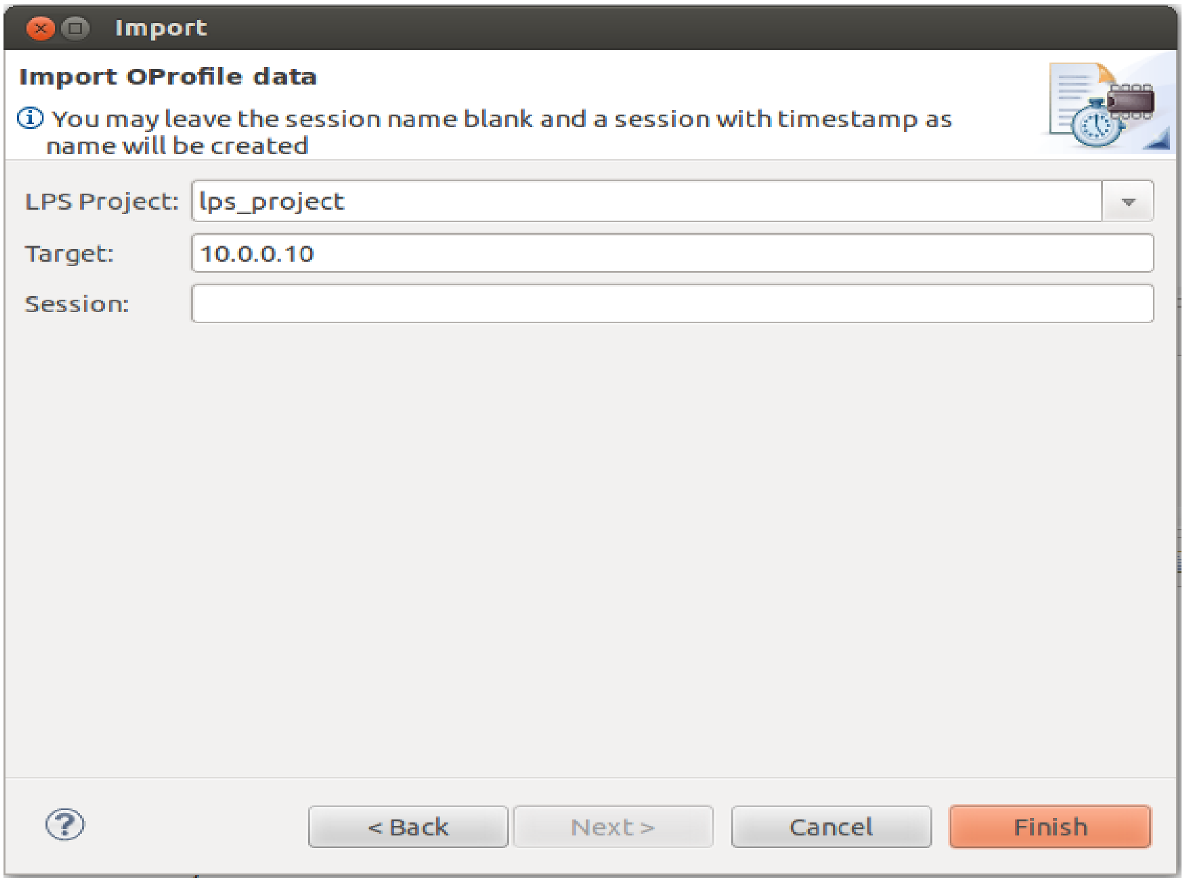
Data Folder on Local System:

CPU Type:

Figure 78: Import Wizard – Selecting OProfile data folder

Select either the Remote or Local radio button to import the OProfile data from a remote target or local file system, click the Browse button to select the data folder. The CPU type is detected and filled in. Click next to continue to the next page as shown in *figure 79.*

You can import the profile data into an existing project or create a new project. The target field is filled in with the remote target / local file system IP address. You may name or leave the session name blank. The timestamp is used if the session name is left blank.



Import OProfile data

You may leave the session name blank and a session with timestamp as name will be created

LPS Project: lps_project

Target: 10.0.0.10

Session:

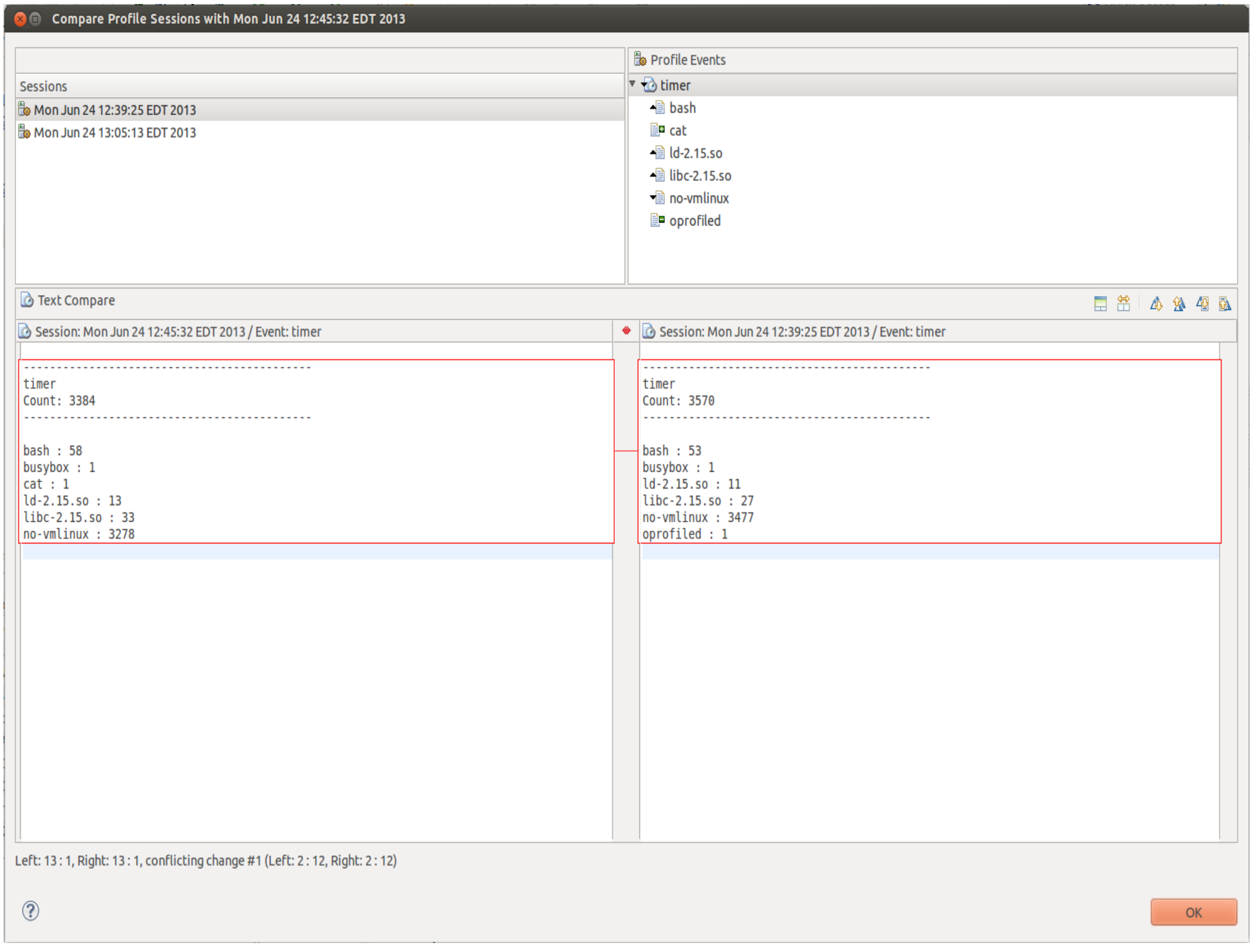
Figure 79: Import Wizard – Selecting LPS project and session

3. Click Finish to import OProfile data. The new session creation and import progress appears in the 'Profiling Monitor' view. Select this session and right-click to analyze or to view as described in previous sections (Analyzing the Profiled Data).

Comparing Profiling Sessions

During development you will profile your application and analyze the data and fix the performance bottlenecks. You will profile the application again. You can compare profile sessions in the same target by using the profile session comparator included in TimeStorm. Select the more recent session with modified code in the 'Profiling Monitor' view, right-click in the view, and select 'Compare'. The

‘Compare Profile Sessions’ window opens with other sessions. Select and double-click the other session you want to compare to. The ‘Compare Profile Sessions’ window compares and displays the profiling results side by side as *shown in figure 80*.


Compare Profile Sessions with Mon Jun 24 12:45:32 EDT 2013
Sessions
Mon Jun 24 12:39:25 EDT 2013
Mon Jun 24 13:05:13 EDT 2013
Profile Events
timer
bash
cat
ld-2.15.so
libc-2.15.so
no-vmlinux
oprofiled
Text Compare
Session: Mon Jun 24 12:45:32 EDT 2013 / Event: timer
Session: Mon Jun 24 12:39:25 EDT 2013 / Event: timer
OK

```
--------------------------------------------
timer
Count: 3384
--------------------------------------------

bash : 58
busybox : 1
cat : 1
ld-2.15.so : 13
libc-2.15.so : 33
no-vmlinux : 3278
```

```
--------------------------------------------
timer
Count: 3570
--------------------------------------------

bash : 53
busybox : 1
ld-2.15.so : 11
libc-2.15.so : 27
no-vmlinux : 3477
oprofiled : 1
```

Left: 13 : 1, Right: 13 : 1, conflicting change #1 (Left: 2 : 12, Right: 2 : 12)

Figure 80: LPS – Comparing two sessions side-by-side

The profile events and the application / library names are displayed in the ‘Profile Events’ pane. You can double click on the name to just display the profile count for that application.

Valgrind Remote Support

TimeStorm adds Remote target support to the Valgrind tools from Eclipse.

Valgrind is an Instrumentation framework for building dynamic analysis tools that can be used to profile applications. Valgrind tools are used to detect memory management and threading problems.

Note: To profile using Valgrind, you should have valgrind 3.3.0 (or later) installed on your host and target.

Profiling using Valgrind

Refer to http://wiki.eclipse.org/Linux_Tools_Project/Valgrind/User_Guide for information of using Valgrind.

To profile on a remote target,

1. right click on the project in the project explorer, select Profiling Tools > Profiling Tools Configurations as shown in *figure 81.*

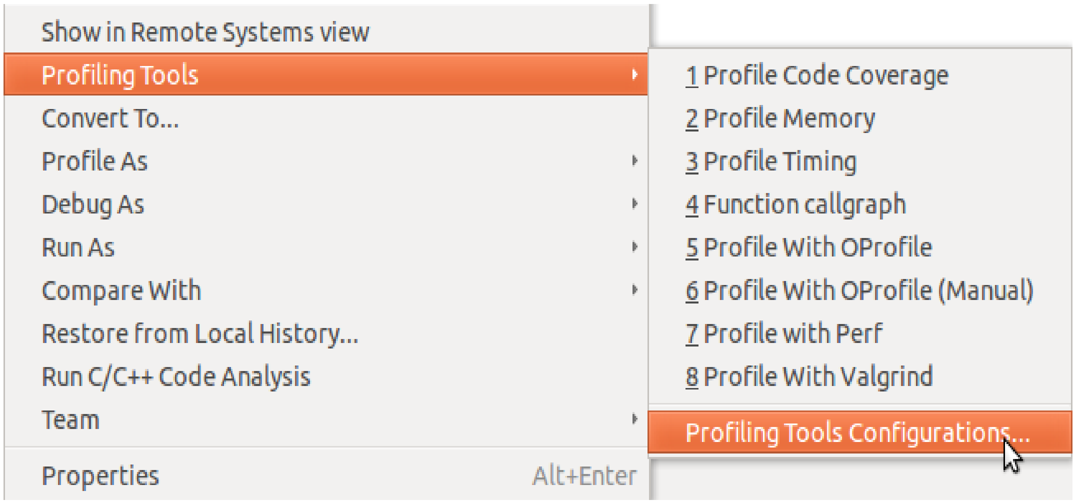
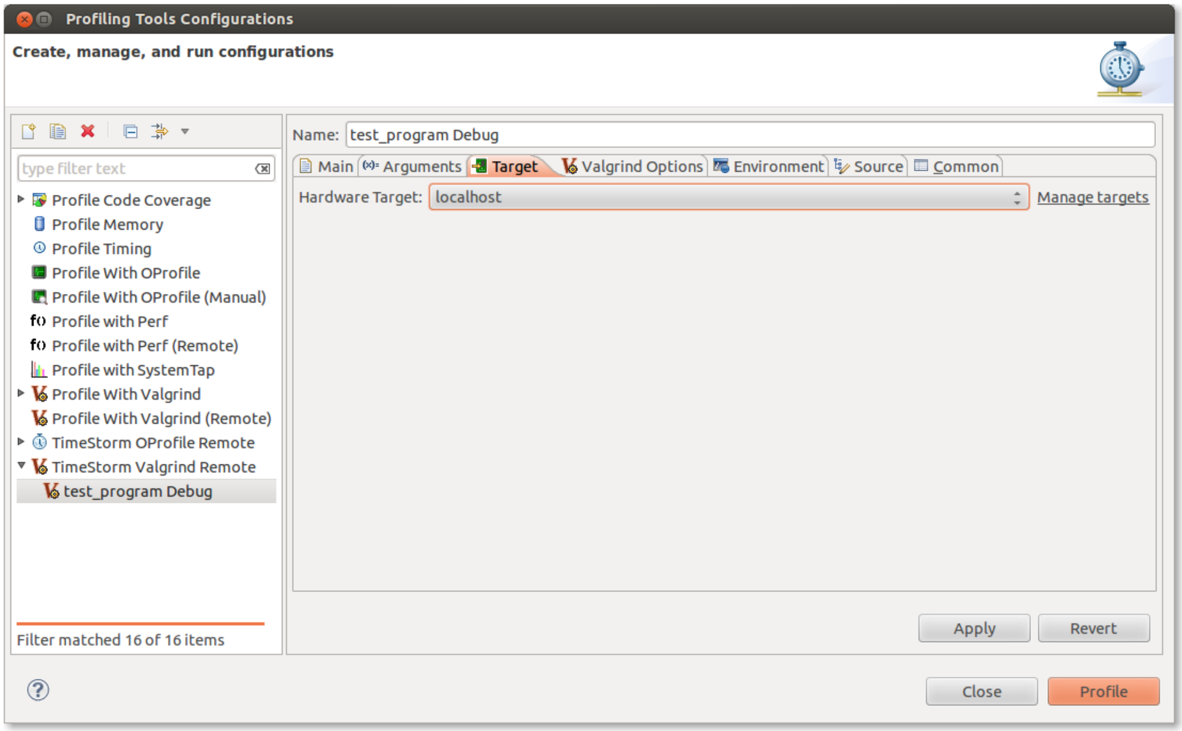


Figure 81: Launching Profile Tools Configuration

2. Select 'TimeStorm Valgrind Remote'.
3. Click New launch configuration icon in top left corner. This will create a new profile launch configuration and the launch configuration tabs are displayed in the right panel (shown in *figure 82)*

4. The launch configuration dialog has tabs as explained in http://wiki.eclipse.org/Linux_Tools_Project/Valgrind/User_Guide and an additional target tab as shown in *figure 82* to select the remote target.



Profiling Tools Configurations

Create, manage, and run configurations

Name: test_program Debug

Hardware Target: localhost

Figure 82: Creating Valgrind profile launch configuration

After you have finished configuring the settings, click the Profile button to start profiling.

LTTng

Eclipse LTTng tools bundled with TimeStorm use ssh connection to communicate to targets and hence they work with remote targets too.

Please note that LTTng tools are not integrated with the hardware targets defined in TimeStorm. You have to define the target IP address, username and password for LTTng to communicate to the target.

For help on how to use LTTng tools, please refer to
http://wiki.eclipse.org/Linux_Tools_Project/LTTng2/User_Guide

Advanced Topics

Building from the Command Line

TimeStorm has been designed so that projects can be built from the command-line as well as the IDE. This feature exists so that customers with build systems can easily integrate projects created with TimeStorm into their automated build system.

As part of the build process, TimeStorm scans the current project and builds a list of files in the project as well as the dependencies on other files in the project. TimeStorm then uses that dependency list, along with the information in the build configuration, to create several GNU make files. TimeStorm then builds the project by executing make with the generated files to perform the build steps. Each build configuration creates a make file in a directory named after the build configuration under the project directory.

For example:

```
<workspace>/my-project/Debug/
                           makefile
<workspace>/my-project/Profile/
                           makefile
```

The make file overrides the following standard variables to control what tools are used in the standard make build rules.

```
RANLIB
CPP
AS
AR
OBJCOPY
DEBUGGER
STRIP
OBJDUMP
CC
NM
CXX
LD
```

Each of these tools is changed to use the location the tool chain of the development host. If the location is different on the build machine, one of the following strategies followed:

Create symlinks — This is the easiest path. On the build machine, create symbolic links to the tools in the location where the make file expects them to be. Going this route introduces a configuration dependency on the build machine, so this should documented by the build team, and, in the best of situations, a script should be created to automate the creation of the links.

Set environment variables — With make, if an environment variable is set, it will not be overridden by a make variable. The script that runs the make file will need to set all of these variables to the right location before invoking make.

Auxiliary Files

TimeStorm will over-write the make file each time is performs a build. In order to allow for maximum customization, TimeStorm builds the make file with several optionally included files. Optionally included files are incorporated into the make file if they exist; otherwise, if the file does not exist, the reference to the file is ignored.

TimeStorm optionally includes these files:

> **…/makefile.init** — This file is called at the beginning of the makefile. It can be used to customize initialization.
>
> **../makefile.defs** — This file is called after initialization, but before objects are compiled. It can be used to supply custom macro definitions.
>
> **../makefile.targets** — This file is called at the end of the makefile. It can be used to supply customized target information.

Notice TimeStorm will look for the files in the parent of the build configuration directory. This makes it easy to share make file customizations across build configurations.

Using Source Code Control from the Command-line

Some organizations use a source code control (SCC) system that can only be used from the command-line or some other external tool. TimeStorm can be used with these tools as long as care is taken in what files are placed under source control and other configuration measures are taken. This section lists out the files and directories that should not be placed under SCC and other per-file configuration information.

File or Directory/	Notes
Workspace	
<workspace>/.metadata	This directory contains state information, such as window positioning, for the current workspace. Exclude this entire directory from SCC management.
C / C++ Projects	
<workspace>/<c project>/.cproject <workspace>/<c project>/.products <workspace>/<c project>/.project	These files contain project-oriented information, such as what appears in the properties dialog for a project. Add all of these files into source control.

About Timesys

Timesys is the provider of LinuxLink, a high-productivity software development framework that dramatically simplifies and speeds up embedded Linux application development. The LinuxLink framework includes the Linux kernel, cross-toolchain, application development IDE, an award winning build system called Factory, a vast library of middleware packages, software stacks and libraries, documentation and expert technical support. LinuxLink enables development teams to consistently build and maintain a custom, open source embedded Linux platform through regularly updated Linux sources, proven middleware packages, and a scriptable GNU-based build environment. LinuxLink reduces the time, resources, risk and cost associated with building a product based on open source Linux. For more information, visit: www.timesys.com.

Timesys Corporation
428 Forbes Avenue
Pittsburgh, PA 15219
+1 412 232 3250
+1 866 392 4897
Fax: +1 412 232 9818